RESERVOIR-COMPUTING-BASED, BIOLOGICALLY INSPIRED ARTIFICIAL NEURAL NETWORKS AND THEIR APPLICATIONS IN POWER SYSTEMS

A Dissertation
Presented to
The Academic Faculty

by

Jing Dai

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2013

RESERVOIR-COMPUTING-BASED, BIOLOGICALLY INSPIRED ARTIFICIAL NEURAL NETWORKS AND THEIR APPLICATIONS IN POWER SYSTEMS

Approved by:

Dr. Ronald G. Harley, Advisor
School of ECE
Georgia Institute of Technology

Dr. Steve M. Potter
School of BME
Georgia Institute of Technology

Dr. Thomas G. Habetler
School of ECE
Georgia Institute of Technology

Dr. Ganesh Kumar Venayagamoorthy
Holcombe Department of ECE
Clemson University

Dr. Jennifer E. Michaels
School of ECE
Georgia Institute of Technology

Date Approved: March 11, 2013

To my dear parents

Mr. Bayi Dai
Mrs. Hongyan Zhou

my dear husband

Dr. Pinjia Zhang

And our first child, Tyler Dai Zhang.

ACKNOWLEDGEMENTS

A doctoral dissertation is usually considered to be a personal accomplishment. However, it would not have been possible for me to finish this work without the inspiration, encouragement and support from many people.

First of all, I would like to express my most sincere gratitude to Dr. Ronald G. Harley. He has been a wise and trustful advisor throughout the entire process. He is always supportive, kind and thoughtful, especially when I have doubts about myself. He is like a father to me. I never could have come this far without his continuous support. I am deeply grateful for his guidance.

I am grateful to Dr. Thomas G. Habetler for his knowledge and experience in my interactions with him, and especially for his suggestions and guidance on my research work. I would like to thank Dr. Jennifer E. Michaels for being my dissertation committee member and for her support in completion of this work.

I would like to acknowledge the National Science Foundation (NSF) of the United States for providing financial support for the research project EFRI #1238097, which is entitled "EFRI-COPN: Neuroscience and Neural Networks for Engineering the Future Intelligent Electric Power Grid". Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

I would like to especially thank Dr. Ganesh Kumar Venayagamoorthy from Clemson University for his continuous support and encouragement. He offered me the great opportunity to join this fascinating, inter-disciplinary research project. I would also like to thank Dr. Steve M. Potter from the School of Biomedical Engineering in Georgia

Institute of Technology for his valuable inputs and constant support, especially in the field of neuroscience.

Also a thank you goes to Dr. Joy Mazumdar. His work on power system nonlinear load modeling using Echo state network has been the basis for this work and has inspired me in my research.

It is very fortunate for me to have the opportunity to work with the exceptional fellow graduate students in the electric power group at the Georgia Institute of Technology. Among them, I wish to especially thank Dr. Jiaqi Liang, Diogenes Molina, Yi Du, Liang Du, Dr. Yao Duan, Dr. Yi Yang, Dr. Siwei Cheng, Dr. Wei Qiao, Dustin Howard, Dr. Anish Prasai, Dr. Jyoti Sastry, Frank Kreikebaum. Andrew Paquette, Prasad Kandula, Dr. Debrup Das, Jorge Hernandez, Dr. Rohit Moghe, Dr. Sangtaek Han. Dr. Stefan Grubic, Yi Deng, Lijun He, Jie Dang, Dawei He for the invaluable collaborations and discussions.

There are numerous names of faculty, family and friends that I should mention here, who have helped me during my years at Georgia Tech. I want to express my gratitude to all of the people I know.

Last but not least, I owe the greatest debt of gratitude to my husband, Dr. Pinjia Zhang. No words in the whole world can express how much his support has meant to me throughout these years. He is always there, encouraging me, supporting me during every single step in this long journey. I also want to thank my parents. Their unconditional love has always been the source of my power. They never asked for anything from me; they always had faith in me and respected my choice of pursuing a Ph.D. I will become a mother in April. Our son is a miracle and the best thing that ever happened to me. I

become stronger because of him. Six years is a long time in a person's entire life. It feels even longer when you are always searching for answers to difficult problems in your research. I am so lucky to have my family. I know that I am never alone on this rough road. The past six years will always be the most precious memory in my life.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

ACD	Adaptive critic design
AD	Action-dependent
ANN	Artificial neural network
APF	Active power filter
BIANN	Biologically-inpired artificial neural network
CI	Computational intelligence
CONVC	Conventional controller
DHP	Dual heuristic dynamic programming
DR	Dynamic reservoir
DIFG	Doubly-fed induction generator
DSP	Digial signal process
ESN	Echo state network
FACTS	Flexible-AC-transmission-system
GA	Generic algorithm
GDHP	Globalized dual heuristic dynamic programming
HDP	heuristic dynamic programming
HRF	Harmonic reference frame
I&F	Integral and fire
I&FB	Integral-and-fire-or-burst
LNN	Living neural network
LSM	Liquid state machine
MEA	Multi-electrode array
MFR	Mean firing rate
MLP	Multip-layer perceptron
MSE	Mean squared error
PCC	Point of common coupling
PWM	Pulse-width modulation
PI	Proportional-integral
PID	Proportional-integral-derivative

PRBS	Pseudorandom binary sequence
RBF	Radial basis function
RNN	Recurrent Neural Network
RTDS	Real-time digial simulator
SNN	Spiking neural network
STDP	Spike-timing-dependent plasticity
STATCOM	Static sychronous compensator
TDL	Time-delayed loop
THD	Total harmonic distortion
VSLI	Viltage stability load index
WAM	Wide-area monitor
WTG	Wind turbine generator

SUMMARY

Computational intelligence techniques, such as artificial neural networks (ANNs), have been widely used to improve the performance of power system monitoring and control. Although inspired by the neurons in the brain, ANNs are largely different from living neuron networks (LNNs) in many aspects. Due to the oversimplification, the huge computational potential of LNNs cannot be realized by ANNs. Therefore, a more brain-like artificial neural network is highly desired to bridge the gap between ANNs and LNNs.

The focus of this research is to develop a biologically inspired artificial neural network (BIANN), which is not only biologically meaningful, but also computationally powerful. The BIANN can serve as a novel computational intelligence tool in monitoring, modeling and control of the power systems.

A comprehensive survey of ANNs' applications in power system is presented. It is shown that novel types of reservoir-computing-based ANNs, such as echo state networks (ESNs) and liquid state machines (LSMs), have stronger modeling capability than conventional ANNs. The feasibility of using ESNs as modeling and control tools is further investigated in two specific power system applications, namely, power system nonlinear load modeling for true load harmonic prediction and the closed-loop control of active filters for power quality assessment and enhancement. It is shown that in both applications, ESNs are capable of providing satisfactory performances with low computational requirements.

A novel, more brain-like artificial neural network, i.e. biologically inspired artificial neural network (BIANN), is proposed in this dissertation to bridge the gap between

ANNs and LNNs and provide a novel tool for monitoring and control in power systems. A comprehensive survey of the spiking models of living neurons as well as the coding approaches is presented to review the state-of-the-art in BIANN research. The proposed BIANNs are based on spiking models of living neurons with adoption of reservoir-computing approaches. It is shown that the proposed BIANNs have strong modeling capability and low computational requirements, which makes it a perfect candidate for online monitoring and control applications in power systems.

BIANN-based modeling and control techniques are also proposed for power system applications. The proposed modeling and control schemes are validated for the modeling and control of a generator in a single-machine infinite-bus system under various operating conditions and disturbances. It is shown that the proposed BIANN-based technique can provide better control of the power system to enhance its reliability and tolerance to disturbances.

To sum up, a novel, more brain-like artificial neural network, i.e. biologically inspired artificial neural network (BIANN), is proposed in this dissertation to bridge the gap between ANNs and LNNs and provide a novel tool for monitoring and control in power systems. It is clearly shown that the proposed BIANN-based modeling and control schemes can provide faster and more accurate control for power system applications.

The conclusions, the recommendations for future research, as well as the major contributions of this research are presented at the end.

CHAPTER 1 INTRODUCTION AND OBJECTIVE OF PROPOSED RESEARCH

1.1 Background of Proposed Research

Worldwide concern about the environmental pollution and a possible energy crisis has led to increasing interests in innovative technologies for generation of clean and renewable electrical energy. All the emerging technologies have brought new challenges to the power quality and power system operation and control. For instance, high penetration of wind and solar energy has led to severe challenges to power system control and operation, due to their unpredictable nature and limited energy storage capability in the system. In addition, with the increasing use of power electronics devices, maintaining the power quality of the power system has been more and more challenging due to harmonic pollution caused by power switches. All these emerging technologies are calling for more advanced monitoring, control and operation technologies for the power system.

A power system is highly nonlinear due to numerous nonlinear devices in it (e.g. switches in power electronic devices), minor and large disturbances or faults at different levels (e.g. mechanical or electrical faulted generators), changes in load conditions over time, transmission line tripping, and growth of renewable energy sources in recent years (e.g. wind and solar farms). It is extremely difficult to model all the rich dynamics in the power system using explicit mathematical equations. Conventional control and monitoring methods, such as linear control, sometimes are ineffective for controlling power systems under variable operating conditions due to their nonlinear and dynamic nature. Computational intelligence (CI) techniques offer a solution to this problem and

have been introduced over the past few decades to improve the performance of power system monitoring and control.

CI techniques, such as fuzzy systems, artificial neural networks (ANNs), generic algorithms (GA), have drawn much attention and have been widely used for optimization, monitoring and control of complex systems. Among these techniques, ANNs have shown exceptional capability of modeling and controlling complex systems because of their intrinsic nonlinearity and computational simplicity. A significant amount of research has been done on ANN-based modeling and control since the 1970s. ANNs are inspired by the neurons in the brain, which have great computational power. Although ANNs were first proposed to mimic the behavior of a group of living neurons, they are largely different from living neuron networks (LNNs) in many aspects. ANNs are only an over-simplified model of the LNNs. The format of the information conveyed between neurons, network topology and configuration, and the learning mechanisms are all fundamentally different between ANNs and LNNs. Due to the oversimplification, the huge computational potential of LNNs has not yet been realized by ANNs. In recent years, the need of developing a more brain-like artificial neuron network to better explore the computational potential of LNNs has drawn more and more attention. The development of such artificial neuron networks can greatly assist the study of actual LNNs, improve the computational capability of ANNs, and eventually provide a scheme of using real living neuron networks for monitoring and control of complex systems.

The main focus of this research is to develop a biologically inspired artificial neural network (BIANN), which is not only biologically meaningful, but also computationally

powerful. The BIANN will serve as a novel computational intelligence tool in monitoring, modeling and control of the power systems.

1.2 Artificial Neural Networks (ANN)

An ANN is a computational structure which consists of simple processing units called neurons (also referred to as nodes) connected to each other. It is this interconnected group of artificial neurons that uses a mathematical or computational model for information processing. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network. Each neuron receives input signals via a series of connections, coming from either other neurons or external signals. Each connection has a weight associated with it. In more practical terms, artificial neural networks are nonlinear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data [1].

1.2.1 Single Artificial Neuron

In an ANN, each artificial neuron (or node) has an op-amp summation type of structure. Each input signal (continuous variable or discrete pulses) flows through a gain or weight (called synaptic weight or connection strength) which can be positive, zero or negative. Figure 1.1 illustrates a typical artificial neuron, with connection weights of w_1, w_2, w_3 and w_4 respectively. If signal values of x_1, x_2, x_3 and x_4 are placed on the inputs to this node, as illustrated, the total weighted sum S of the neuron is $S = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$.

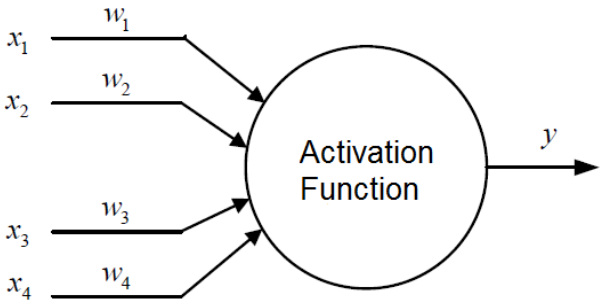
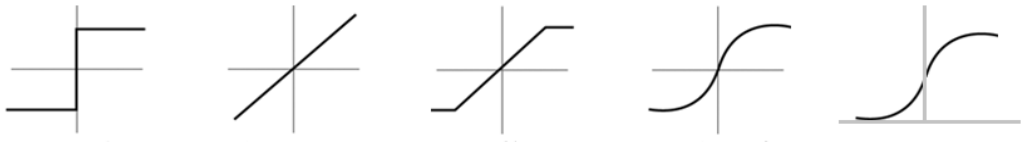


Figure 1.1: A typical artificial neuron.

Several commonly used types of activation functions used in artificial neurons are shown in Figure 1.2. These are defined, respectively, as step, linear (bipolar), threshold, hyperbolic tangent (or tan-sigmoid) and sigmoid (or log-sigmoid) functions. The magnitudes of these functions vary between 0 and 1, or -1 to +1 as indicated. The linear function can be unipolar or bipolar.



(a)Step (b) Linear (c) Threshold (d) Hyperbolic tangent (e) Sigmoid

Figure 1.2: Commonly used activation functions of the artificial neurons.

The computation performed by any single artificial neuron is simple, but complex behavior emerges when neurons are connected to each other, forming a network. Instead of having to explicitly program them with a particular data processing algorithm, ANNs have the ability to learn from the data presented to them. Learning in a neural network takes place by the modification of the connection weights between nodes.

1.2.2 ANN Structures

Many ANN topologies have been proposed over the past few decades. The ANNs are generally classified as feedforward and recurrent types [1]. In the feedforward class, the signals flow only in the forward direction, whereas in recurrent neural network (RNN), the signals can flow in the forward as well as the backward or the lateral direction. An ANN can be defined as static or dynamic, depending on whether it emulates static or dynamic system. An ANN is characterized by its input-output mapping property. For static mapping, the feed-forward ANNs are preferred, whereas for dynamic or temporal mapping, the RNNs are preferred. A few of the neural network architectures explained by Haykin [1], as well as some more recent ones are listed as follows.

A. Feedforward networks:

- Perceptron [2]
- Adaline and Madaline [3]
- Multilayer Perceptron Neural Network [4]
- Radial Basis Function Network [5]
- General Regression Network [6]
- Self-Organizing Feature Map [7]
- Learning Vector Quantization Network [8]
- Probabilistic Network [9]

B. Recurrent networks:

- Elman Network and Jordan Network [10]
- Echo state network [11]
- Hopfield Network [12]

- Adaptive Resonance Theory Network [13]
- Bi-Directional Associative Memory Network [14]

The characteristic of a neural network is that, when trained correctly, the mapping formed by the network can exhibit some capability for generalization beyond the training data. Moreover, typical to neural networks and associated learning algorithms, is that they are somewhat robust against redundant input variables or missing values in the training data.

1.2.3 ANN Learning Mechanisms

Any neural network requires a learning algorithm to identify or approximate a process. There are three basic classes of learning: supervised, unsupervised and reinforcement learning.

In *supervised learning* [15], the process starts with teacher data (or training data from the process or plant) which represent examples of the desired model behavior. The teacher data is used to train an ANN such that the ANN approximates (fits) the teacher data. When the trained ANN then receives an input sequence, it should generate an output which resembles the actual output of the original system if it were supplied by the same input. A commonly used cost function is the mean-squared error which tries to minimize the average error between the ANN outputs and the target values. Minimizing this cost function using gradient descent methods for the class of neural networks called Multilayer Perceptrons (MLPs), results in the well-known backpropagation algorithm for training neural networks [16]. Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation).

In contrast to supervised learning, the objective of *unsupervised learning* [17] is to discover patterns or features in the input data with no help from a teacher, basically performing a clustering of the input space. In unsupervised learning, given some data x, the cost function to be minimized can be any function of the data x and the ANN output $g(x)$ .The cost function is dependent on the task and certain priori assumptions (the implicit properties of the model, its parameters and the observed variables).

In *reinforcement learning* [18], a teacher, though available, does not present the expected answer but only indicates if the computed output is correct or incorrect. The information provided helps the network in its learning process. A reward is given for a correct answer computed and a penalty for a wrong answer.

1.3 Living Neural Networks (LNNs)

Brains, which consist of huge amounts of living neurons, are exquisitely good at adaptive real-time interaction with the world. How do neural systems adapt and learn to be highly effective at decision-making? A few research groups are approaching this question by studying small LNNs in culture [19]. These LNNs have far less uncontrolled parameters than intact brains, hence are much more amenable to detailed observation and manipulation. By investigating LNNs, a better understanding can be gained of how single-neuron activity combines to form the network-level processing that takes in sensory input, stores memories, and controls behavior.

1.3.1 Use Multi-electrode Array (MEA) to Study LNNs

Recent developments in MEA technology enable researchers to conduct various research activities on LNNs. The standard type of in-vitro MEA comes in a pattern of 8×8 or 6×10 electrodes. Electrodes typically have diameters between 10 and 30 μm.

Networks of neurons are grown in a culture, using multi-electrode culture dishes. LNNs can be maintained in-vitro on dishes for several months. Figure 1.3 shows a week-old culture of ~50 000 neurons and glia from an embryonic rat cortex, growing in a MEA and forming a dense network of 1-2 mm across. Fifty-nine 30 μm electrodes spaced at 200 μm intervals connect a few hundred of the network's neurons to the outside world [20].

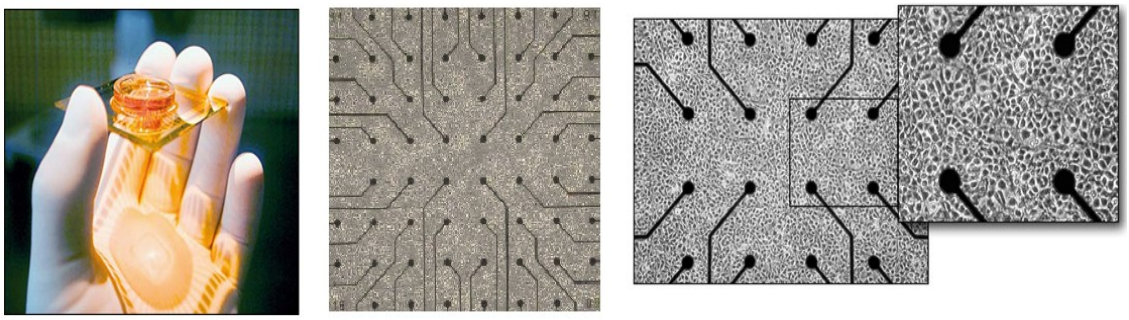


Figure 1.3: MEA with a culture of living neurons growing on top [20].

Electrical stimuli are delivered via the electrodes in complex spatio-temporal patterns, representing sensory input to the network. The output of the network, which are the action potentials (or spikes, 10-100 μV), are produced by neurons on or near the extracellular electrodes, and recorded and processed in real time. Thus, MEAs allow a continuous, two-way interface to living neuron tissue, making it possible to create close-loop hybrid (living + artificial) neuron systems [21].

MEAs have been successfully used in many open-loop or closed-loop applications, for example, to receive artificial sensory input [22], to control robots [23] or control simulated animals [24]. Since the field of such embodied cultured networks is new, many questions remain unanswered about how in-vitro neuronal networks process, store and

use information for real-time control. In addition, all these applications of LNNs have been conducted on very simple tasks. Effective methods of training LNNs are still unknown, which makes it difficult to use LNNs on complex tasks such as monitoring and control of power system.

1.3.2 Fundamental Differences between LNNs and ANNs

Despite the fact that the ANNs were proposed to mimic the behaviors of the real neuronal system, the differences between LNNs and ANNs remain significant. In the process of building a mathematical model for a group of living neurons, many of the important features of the living neuron were omitted by earlier researchers for simplicity. Some of the fundamental differences are listed in Table 1.1.

Table 1.1: Fundamental differences between LNNs and ANNs.

	LNNs	ANNs
Number of neurons used	Large, 10^4 +	Usually small, up to 10^3
Type of neurons	Non-homogeneous, for example, excitatory and inhibitory neurons	Usually share the same activation function
Format of data flow	In the form of spikes	Continuous, analog signal
Latency taken into account?	YES	NO
Have a system clock?	NO	YES
Learning Mechanism	Still unknown	supervised, unsupervised or reinforcement training

One of the major differences between ANNs and LNNs lies in the format of data flow. In ANNs, information is transmitted among neurons as a continuous analog signal; while in LNNs, information is transmitted among neurons as spiking signals, where the real information is encoded in the patterns of these spiking signals. Due to the different

formats of data flow in LNNs and ANNs, ANNs cannot assist understanding of the learning, information processing and storage in LNNs.

To assist the understanding of learning processes in LNNs and provide effective methods to utilize the huge computational potential of LNNs, it is essential to develop a novel type of artificial neural network which can better emulate the behavior of LNNs.

1.4 Filling the Gap between LNNs and ANNs

After decades of research on LNNs, various spiking models of living neurons have been proposed to emulate the various behaviors of living neurons, such as their spiking patterns, and neuronal latencies. However, the key question still remains unanswered: how do LNNs learn, process and store information?

Despite the various neuron models that have been proposed by the scientific community, very little is known on the learning mechanism of LNNs. It is still unclear how the information is encoded in the signals that are transmitted in LNNs, how the learning mechanism is realized, and eventually, how to utilize such huge computational potential for practical applications. It is necessary to develop a new type of artificial neural network, which can better emulate LNNs, with effective training mechanisms. To better mimic the behavior of LNNs, a novel family of artificial neural networks, i.e., biologically inspired artificial neural networks (BIANNs) [25] based on the spiking models of living neurons, show promise. The study of BIANNs can provide important guidance to the study of LNNs for better understanding the learning mechanism of LNNs and utilizing their huge computational potential for practical applications.

1.5 Thesis Outline

Chapter 2 presents a literature review of the previous work on ANN applications in power systems. The concept of reservoir computing is introduced. Applications of the Echo state network, which is a form of reservoir computing since the year 2002, are reviewed in detail.

Chapter 3 summarizes the previous work related to the BIANNs, which includes most commonly used neuronal coding methods and spiking neuron models in the BIANN research area. A comparison of these commonly used neuron models is given in Section 2.3.

In Chapter 4, an ESN-based approach is proposed to predict the true harmonic contributions of nonlinear loads in the power grid. The performances of three different types of ANNs, for the same application are compared, namely MLPs, RNNs and ESNs. The ESNs clearly outperform the other two types of ANNs as shown in Section 4.3 in terms of modeling accuracy and computational requirements.

As a follow on to Chapter 4, Chapter 5 proposes an ESN-based indirect adaptive control scheme for an active filter to eliminate the harmonic currents injected by those nonlinear loads.

With the performances of the ESNs in system modeling and control applications demonstrated, Chapter 6 introduces a novel BIANN architecture based on the idea of reservoir computing. A mean-firing-rate-based coding method is proposed in Chapter 6. With this coding method and the carefully chosen Izhikevich Model as the spiking neuron model, the proposed BIANNs can be trained online for modeling purposes. Simulation results of using the BIANN as a function approximator are given in Chapter 6.

To further investigate the capability of BIANNs for modeling in power system applications, a BIANN-based modeling approach for modeling performance and behavior of generator in a single-machine infinite-based system (SMIB) is presented in Chapter 7. The performance of the BIANN-based modeling scheme is evaluated through simulation under various conditions. It is clearly shown that the proposed BIANN-based modeling scheme for generators can accurately identify the dynamics of the SMIB power system and has the potential to be used for control applications.

With the modeling capability of BIANNs demonstrated, a HDP (heuristic dynamic programming)-based optimal controller using BIANNs is proposed for controlling of generators in a single-machine-infinite-bus (SMIB) system in Chapter 8. The proposed control scheme is illustrated in various operating conditions and for disturbances in the SMIB system. It is shown that the proposed control scheme outperforms conventional linear controller in all various conditions. In Chapter 8, the performance of the HDP-BIANN controller is also compared against the performance of a HDP-MLP controller for the same generator under small and large disturbances. It is shown that the HDP-BIANN controller damps out the oscillations for the terminal voltage and the rotor speed more effectively than the HDP-MLP controller.

Finally, in Chapter 9, the conclusions, contributions and recommendations for future work on applying the BIANNs to applications in power system are discussed in detail.

CHAPTER 2 PREVIOUS RESEARCH ON ARTIFICIAL NEURAL NETWORK (ANN) APPLICATIONS IN POWER SYSTEMS

2.1 Introduction

The electric power industry is currently undergoing an unprecedented reform. One of the most exciting and potentially profitable recent developments is the increasing usage of artificial intelligence techniques. ANNs have been used in a broad range of power system applications, including fault diagnosis, security assessment, load forecasting, economic dispatch and harmonic analysis. ANNs are used for various purposes in these applications, including pattern classification, pattern recognition, optimization, prediction and automatic control. In spite of different structures and training paradigms, all ANN applications are special cases of vector mapping. The application of ANNs in different power system operation and control strategies has led to improved results compared to conventional approaches. This Chapter gives an overview of ANN applications in power systems.

2.2 ANN Applications in Various Power System Subjects

ANNs have been used in a broad range of power system applications. Over 2000 papers have been published in this area for the period 2000 to 2011, and many successful experiments and practical tests have been reported in these papers. ANNs are used in various power system subjects, and a list of these subjects appears below.

• Load forecasting

Accurate models for electric power load forecasting are crucial for power system operation and planning. It helps to make important decisions including generation and load switching. It can be divided into three categories: short-term load forecasting, which

is typically from one hour to one week; medium-term forecasting, which is typically from one week to one year; long-term forecasting, which is more than one year. ANNs have been widely used for load forecasting, especially for short-term load forecasting [26-37], thanks to the strong modeling capability of ANNs and their high immunity to noise in measurements. ANN-based load forecasting techniques are a type of regression methods, which takes many factors into consideration, such as weather conditions, type of day, customer class and so on.

• Fault diagnosis and protection at the device and system level;

For stability of the entire power system, early detection and isolation of failures at both device and system level are critical. Accurate classification of system failures is of great importance to improve system reliability as well as reduce false alarms. ANN-based fault diagnosis and protection techniques have been widely used in such applications due to their strong multi-factor classification capability [38, 39].

• Security assessment;

Security assessment is another important aspect in power system monitoring and operation. The general monitoring and assessment of power system and its components, such as generator stability assessment [40], power system stability monitoring [41-43], thermal monitoring [44-46], harmonic evaluation [47], are critical for system stability and power quality. These publications show that ANNs can outperform conventional techniques in many of these applications.

• Operational planning and economic dispatch;

Another important application of ANNs in power is in operational planning, such as economic dispatch [48], optimal power flow [49], adaptive load shedding [50, 51], and

power system stabilization enhancement [52, 53]. These techniques can help to optimize the operation of a power system with certain given constraints.

• Modeling, identification and optimal control for power system flexible AC transmission devices.

Flexible AC transmission (FACTs) devices are more and more used in power systems for assisting operation of power systems, power quality enhancement and improving system stability. Due to the complexity and dynamic characteristics of power system, the control of such devices is challenging especially when the system is undergoing some disturbance. ANN-based modeling and control of these devices, for instance, the static synchronous compensator (STATCOM) [54] and power system stabilizer (PSS) [55-58], can improve the performance of these devices under various operation conditions.

• Renewable power generation

With the high penetration of renewable energy, the operation and control of these new types of power generations is critical. Various ANN-based approaches have been proposed in [59-66] to provide such functionality through modeling, generation prediction, stability analysis and control. Over 90% of the publications in this area use traditional ANN models, such as the well-known MLPs, RNNs and RBFs.

2.3 Reservoir-Computing-based Advanced ANNs for Power System Applications

The concept of "reservoir computing" was proposed by Jaeger [67] and Maass [68], independently in 2002. Although Jaeger's ESN approach and Maass's Liquid state machine (LSM) architecture use different terminologies, they share a common

framework. Typically an input signal is fed into a large, random dynamical system (i.e. the "reservoir" or the "liquid") and the dynamics of the reservoir map the input to a higher dimension. Then a simple readout mechanism is trained to read the state of the reservoir and map it to the desired output. The main benefit is that the training is performed only at the readout stage, which requires much lower computational effort than conventional ANNs.

ESNs and LSMs are still recurrent neural networks, only in a special form. Therefore, in theory they can be applied to all the topics that were summarized in Section 2.1. Some researchers have already conducted some studies on applying ESNs or LSMs to the power system. A literature review on ESN applications in power system is given below.

2.3.1 ESNs for Wind Speed Forecasting

ESN-based real-time wind speed estimation for wind power generation was proposed in [60]. Wind turbine generators (WTGs) are usually equipped with one or more well-calibrated anemometers to measure wind speed for system monitoring, control, and protection. The use of these mechanical sensors increases the cost and hardware complexity and reduces the reliability of the WTG system. An ESN was developed [60] to provide a nonlinear inverse model of the WTG dynamics, and to estimate the wind speed in real time from the measured WTG output of electrical power, shaft speed, and blade pitch angle. The estimated wind speed was then used for wind-speed-sensorless control of the WTG system. This removed the need of mechanical sensors to measure wind speed.

The electrical power generation from wind energy to electric energy was represented in [60] by a diagram like that given in Figure 2.1. The conversion of wind energy (P_w) to

turbine mechanical power (P_m) was represented by a wind turbine aerodynamic model. In this model, the turbine mechanical power P_m was represented as a nonlinear function of the wind speed v_w, turbine shaft speed ω_t; and blade pitch angle β. The dynamics of the WTG shaft system was represented by a set of differential equations which transfer the mechanical power from the turbine to the generator (P_{in}). The generator converted the (P_{in}) mechanical power into electrical power (P_e). The losses (P_{loss}) of the system (referred to the generator side) were accounted for in this model. The output electrical power P_e, of the generator was measured.

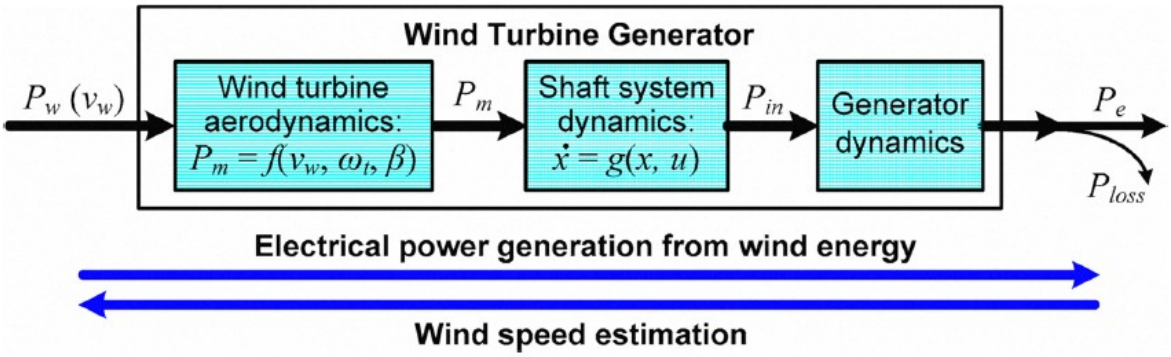


Figure 2.1: The principle of wind speed estimation. [60]

An ESN model was trained as an inverse model from P_e, to v_w, then the wind speed could be estimated from the measured output electrical power, as shown in Figure 2.2.

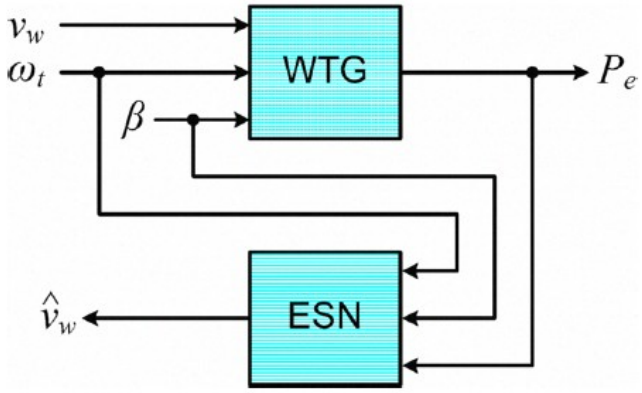


Figure 2.2: Wind speed estimation using an ESN. [60].

The estimated wind speed was then used for wind-speed-sensorless control of the WTG system. The proposed algorithm was verified in [60] by simulation studies. Results showed that the algorithm estimated the wind speed in real time during various wind conditions and the resulting sensorless control strategy provided a precise control for maximum power generation of the WTG system.

2.3.2 ESNs for Online Voltage Stability Load Index Estimation

An ESN-based Voltage Stability Load Index (VSLI) [69] estimation method using data from optimally located phasor measurement units (PMUs) for the IEEE 14-bus test system was proposed in [70]. The VSLI was one of the voltage stability indices which were defined to facilitate the necessary control actions to preclude imminent instability and thereby improve voltage stability in the power system. The VSLI was derived at each load bus using the Thevenin equivalent representation of the system. The system and optimally placed PMUs are shown in Figure 2.3. The ESN used for the VSLI estimation is shown in Figure 2.4.

Three cases for VSLI estimation were considered for the IEEE 14-bus system using ESNs. The first case was the normal operating condition, and the second and third cases were two N-1 contingencies involving the loss of lines 2-3 and line 2-4 respectively. Performance of the ESN for estimating VSLI was evaluated and the results showed high accuracy in the VSLI estimation under normal and disturbed conditions.

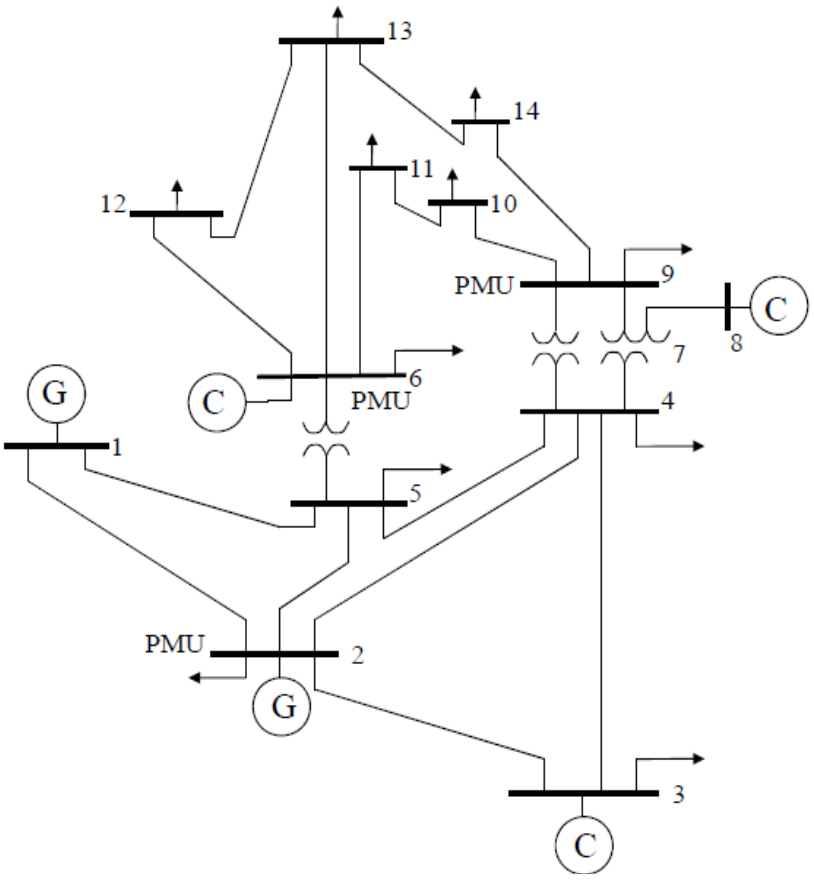


Figure 2.3: IEEE 14-bus test system with three PMUs placed. [69]

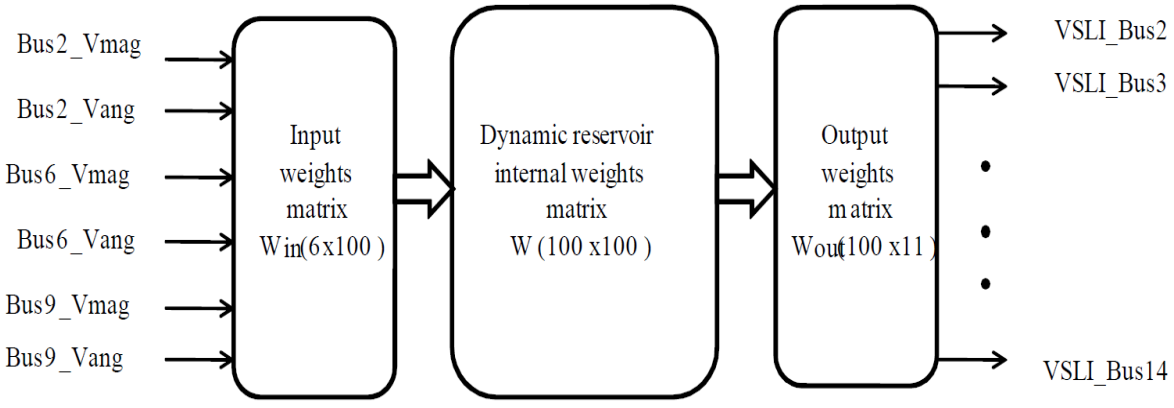


Figure 2.4: ESN for VSLI estimation in power system. [69]

2.3.3 ESNs for Load Forecasting

ESN-based short term load forecasting was proposed in [32, 35]. In [32], such load profiles were fed to a single ESN for all load prediction without considering other factors, such as weather conditions, seasonal variations. In [35] the hourly load data along with only average temperature of each day and day type flag were fed to the ESN and the results showed that the ESN could be trained much easier and with a great deal of accuracy, and this method successfully predicted load demands even using limited input data.

2.3.4 ESNs for Motor Control

An ESN-based direct neural control scheme for a motor in a robot was proposed and validated through experiment in [71]. ESNs were applied to real world data stored during a run of a mobile robot to find a better nonlinear controller then the previously given proportional–integral–derivative (PID) controller.

The considered plant was a two wheeled differential drive mobile robot. The driving wheels of the robot were actuated by two AC-motors. A pulse width modulated (PWM) voltage was generated by the controller. Speed commands from a higher level behavior on a robot notebook were sent to this controller via a serial interface and defined the desired speeds for the implemented classical PID controller algorithm. Actual speed values were sensed and computed from encoders mounted on the motor shafts. The PID controller was based on the assumption that the plant could be modeled as a second order system. However, when the robot needed to cope with frequent fast and non-periodic changes of the desired speeds and applied loads, this simple second order model was not sufficient since it was only a rough approximation of the observed plant dynamics. An

ESN-based control scheme was designed to replace the PID controller shown in Figure 2.5.

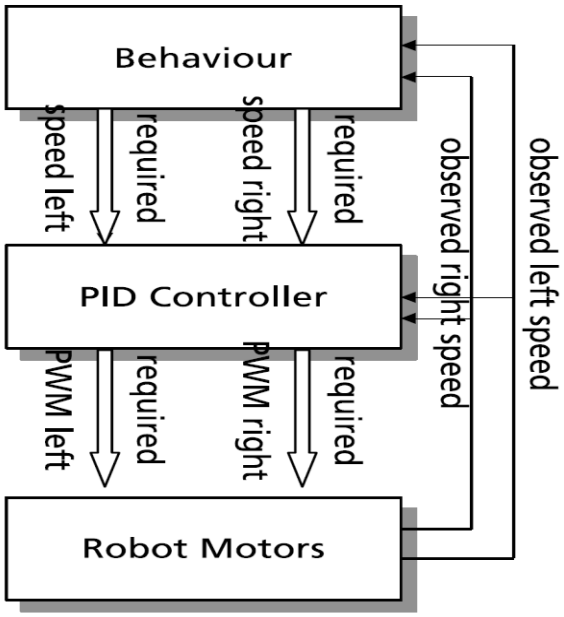


Figure 2.5: Robot motor control system configuration. [71]

The training process of the ESN model and controller had three steps. First, a trace file was generated by a real robot trajectory. It recorded the actual speeds $y(i)$ of the wheels and the PWM duty cycles $u(i)$ over time. Second, the plant model was trained by using the input vectors $u(i)$ for the left and right motor and the observed actual velocities $y(i)$ as teacher. Finally, the "ideal" controller was trained by feeding the inputs with the actual velocities $y(i)$ and the incrementally delayed signals $y(i + dk)$ of those actual velocities, teaching the ESN to follow the non-delayed PWM values $u(i)$. Figure 2.6 shows the two phases of the training.

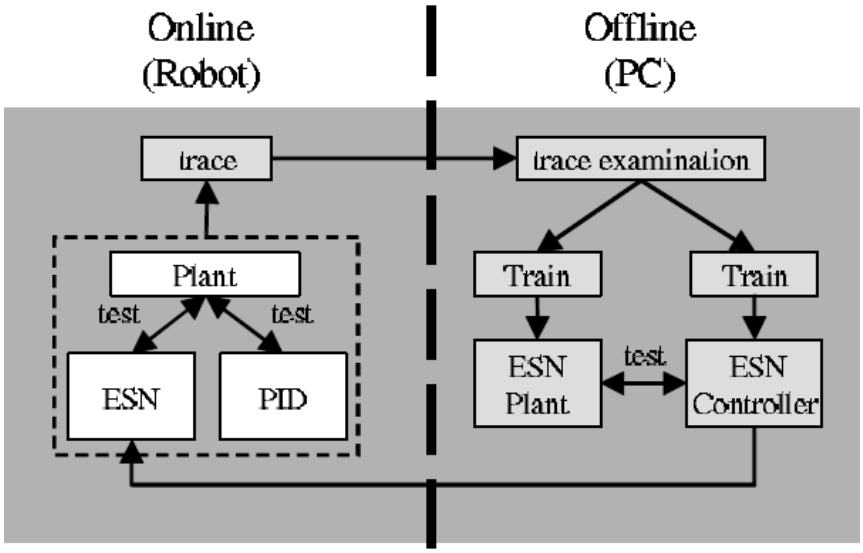


Figure 2.6: ESNs for robot motor control. [71]

Both networks, the ESN model and the ideal ESN controller, were trained in an off-line simulation. A comparison of original PID controller with the ESN controller showed an improvement for the mean absolute error. In addition, investigations on the time behavior of a step-response showed that the actual velocity controlled by an ESN-controller reached its end-value quicker than the PID controller.

2.3.5 ESNs for Thermal Dynamics Identification of Overhead Power Lines

The ESN was proposed to applied to predict the overhead conductor thermal dynamics in real-time in [45]. Due to the inherent non-linearity of overhead conductor thermal behavior, it is usually quite complex to directly calculate the conductor temperature. Therefore the prediction for the conductor dynamic thermal behavior becomes difficult. A well trained ESN model was used to predict the dynamic thermal behavior, and thus to evaluate the dynamic current capacity of the line under current

ambient weather conditions. The thermal dynamic model of the overhead conductors is shown in Figure 2.7.

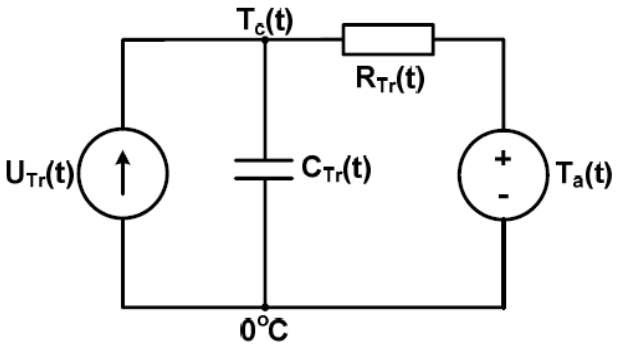


Figure 2.7: Thermal dynamic model of the overhead conductor.[45]

The temperature of the overhead conductor $T_c(t)$ was considered as a nonlinear function of the Ambient Temperature $T_a(t)$ and the Line Current $I(t)$, i.e. $T_c(t) = f(T_a(t), I(t))$.An ESN was trained to learn this nonlinear function using data measured from the overhead conductors in a real power system. The training algorithm of this ESN thermal dynamic identifier is shown in Figure 2.8.

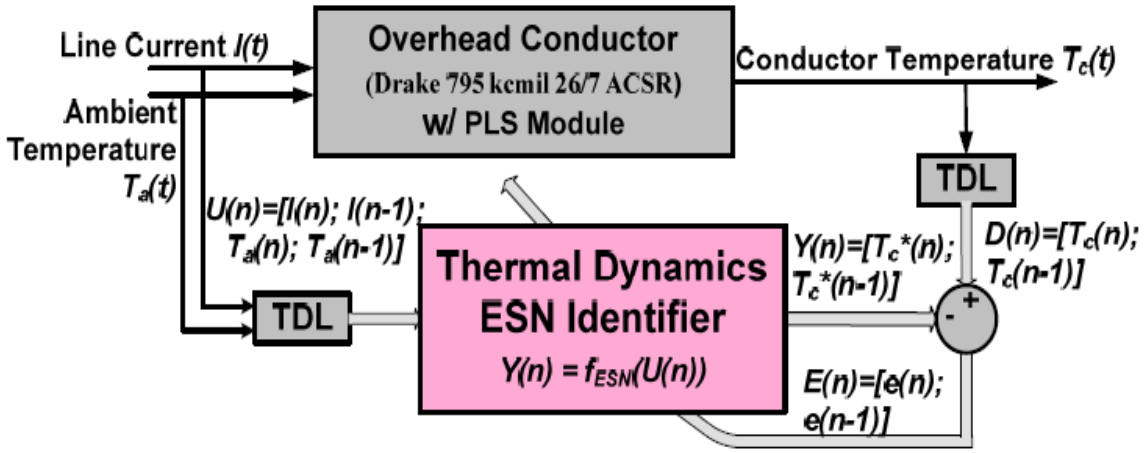


Figure 2.8: Overhead conductor thermal dynamics identifier.[45]

2.3.6 ESNs for Power System Wide Area Monitoring

An ESN was used for the online design of a Wide Area Monitor (WAM) for a multi-machine power system in [72]. A single ESN was used to predict the speed deviations of four generators in two different areas. The performance of this ESN WAM was evaluated for small and large disturbances on the power system. The structure of the multi-machine power system with a WAM is shown in Figure 2.9. The training of the ESN WAM is shown in Figure 2.10.

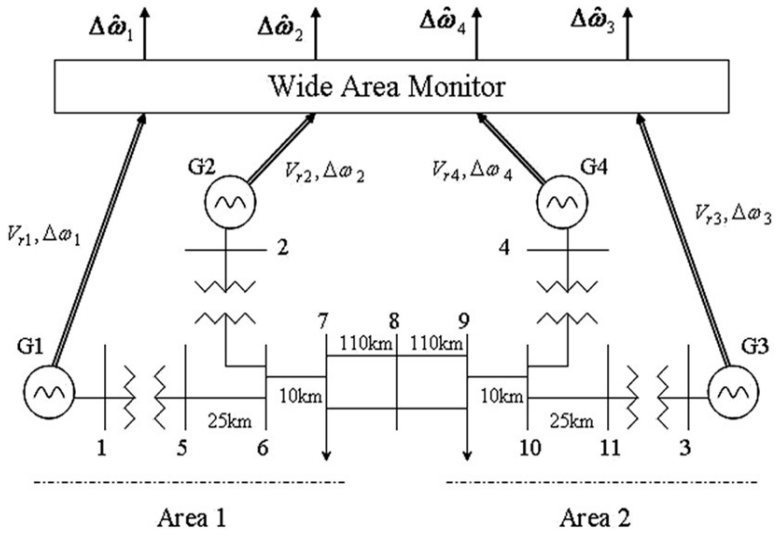


Figure 2.9: WAM predicting the speed deviations of generators. [72]

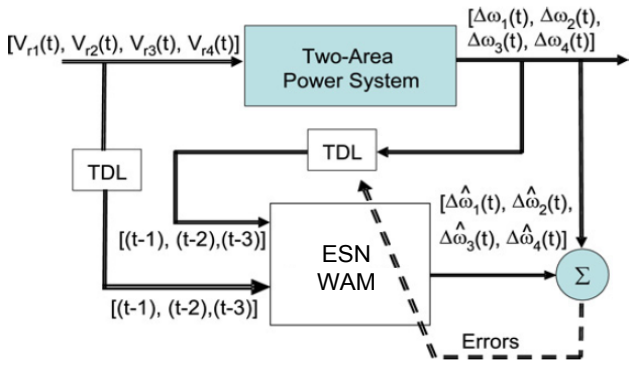


Figure 2.10: Wide area monitor for power system. [72]

2.4 Chapter Summary

In this chapter, a review of ANN applications in power systems has been presented. Conventional ANNs, such as MLP, RBF, haven been widely used in various power system applications, such as load forecasting, fault diagnosis, operation management,

security assessment, control of FACT devices. It has been shown in the literature that ANNs can outperform conventional techniques in many of these applications.

With the proposal of reservoir-computing, a novel type of reservoir-computing-based ANN, the ESN, has shown its advantages in power system nonlinear modeling applications with lower computational effort required. The applications of these novel ANNs in power system have been reviewed in detail. Due to their strong modeling capability and lower computational requirements, these reservoir-computing-based ANNs are ideal for online modeling and control applications for power systems.

The application of these novel ANNs so far has been limited to modeling applications, or simple control application, in which the input and output data of a conventional PID controller was used to train an ANN [71]. Therefore, the potential of these ANNs still need investigation for complex modeling and control applications. In Chapter 4 and Chapter 5, the capability of these ANNs is investigated further for modeling and control in power systems.

CHAPTER 3 PREVIOUS WORKS ON BIOLOGICALLY INSPIRED ARTIFICIAL NEURAL NETWORKS

3.1 Bridge ANNs and LNNs: an Introduction to the BIANNs

The human brain is the best example of known intelligence, with unsurpassed ability for complex, real-time interaction with a dynamic world. ANN researchers trying to imitate the brain's remarkable functionality could benefit by learning more about neuroscience, and the differences between Natural and Artificial Intelligence [73]. In recent years, ANN researchers have been pursuing a more brain-inspired approach, called a Biologically Inspired Artificial Neural Network [25], to bridge the gap between ANNs and LNNs.

Neurons in the brain communicate by short electrical pulses, the so-called action potentials or spikes, so the term "spiking neuron" is often used instead of "neuron" in this research area. Since the goal of the BIANN research is to make the BIANN network more biologically meaningful, all the neurons in the BIANNs should be spiking neurons.

A typical spiking neuron can be divided into three functionally distinct parts, called dendrites, soma and axons. Roughly speaking, the dendrites play the role of the "input device" that collects signals from other spiking neurons and transmits them to the soma. The soma is the "central processing unit" that performs an important nonlinear processing step. If the total input to a neuron exceeds a certain threshold, then an output signal from this neuron is generated. The output signal is taken over by the "output device", the axon, which delivers the signal to other neurons [74]. A typical single neuron is shown in Figure 3.1(a). The junction between two neurons is called a synapse, as shown in Figure 3.1(b). Suppose that a neuron j sends a spike signal across a synapse to a neuron i. It is

common to refer to the sending neuron as the presynaptic neuron and to the receiving neuron as the postsynaptic neuron.

The neuron signals consist of short-duration electrical pulses, which are so-called "action potentials" or "spikes". A spike typically has an amplitude of approximately 100 mV and a duration of 1-2 ms. The shape of each pulse does not change as the action potential propagates it along the axon. A chain of action potentials emitted by a single neuron is called a "spike train" – a sequence of stereotyped events which occur at regular or irregular intervals. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. To a first approximation, it is the number and the timing of spikes which matter [74]. Figure 3.2(a) shows how the noisy voltage signals are sorted to spike trains from recorded neuron output signals. A neuron that emits an action potential is said to "fire". Every time a single neuron fires, the voltage pulse is sorted as a spike. In Figure 3.2(a), the outputs of three neurons are recorded first, then the background noises are filtered out, and only the pulses generated by each neuron are sorted to form its output sequence. An illustration of the signal transmission between interconnected neurons is shown in Figure 3.2(b). The spike trains travel along the axon and are distributed to postsynaptic neurons where they evoke postsynaptic potentials. If a postsynaptic neuron receives several spikes from its pre-synaptic neurons within a short time window, its membrane potential may reach a critical value and an action potential is triggered. This action potential is the output signal, which is transmitted to other neurons.

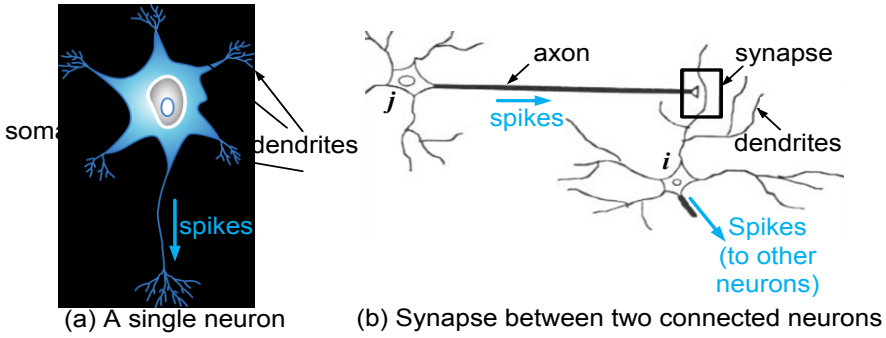


Figure 3.1: Spiking neurons: dendrites, soma, axon, synapse and spike trains [74].

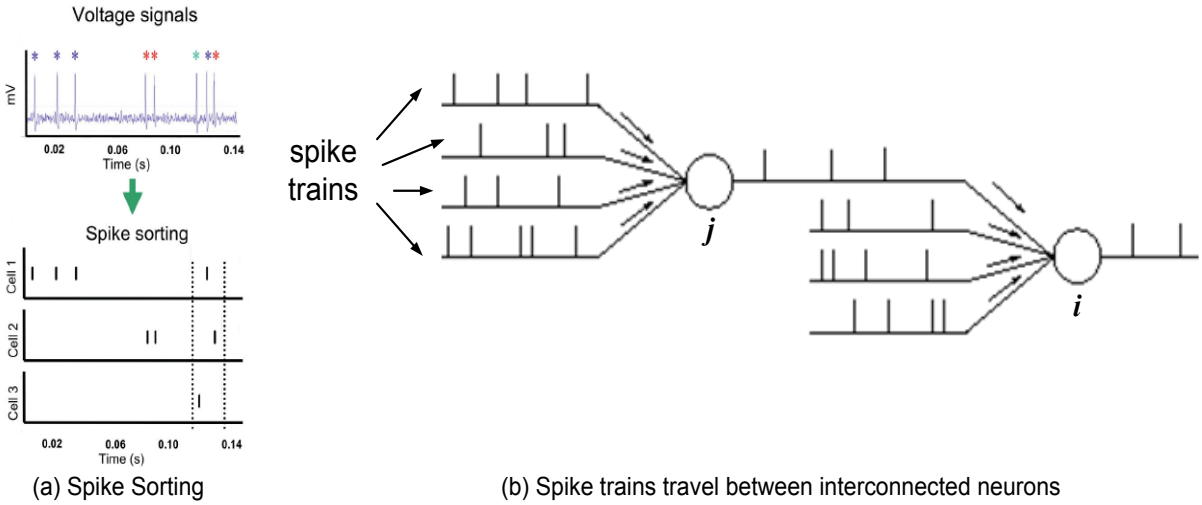


Figure 3.2: Spike trains and their propagation through interconnected neurons [74].

Over the years the research on BIANNs has been mainly focused on three closely related fields: (1) how to encode real world signals into spike trains that can be recognized and processed by spiking neurons? (2) What mathematical models should be used to model a single spiking neuron and a network formed by spiking neurons? (3) As external observers, how to accurately decode the information contained in the spike trains,

in other words, how to read the spikes and understand the message buried in the neuronal activity pattern?

The above questions point to the problems of neuronal encoding, spiking neuron model and neuronal decoding, three fundamental issues in neuroscience. A comprehensive literature review on previous work done by others on these three topics is presented in sections 3.2 and 3.3.

3.2 Neuronal Coding: Encoding and Decoding the Spikes

Neuronal coding, which can be further divided into encoding and decoding, is a long-standing and fundamental issue in computational neuroscience. Encoding refers to transforming real-world signals to spike trains that can be used as stimuli to the neural network, while decoding refers to reading the spike trains out of the neural network and converting them back into meaningful real-world signals [75-78]. An illustration of the encoding and decoding process is shown in Figure 3.3.

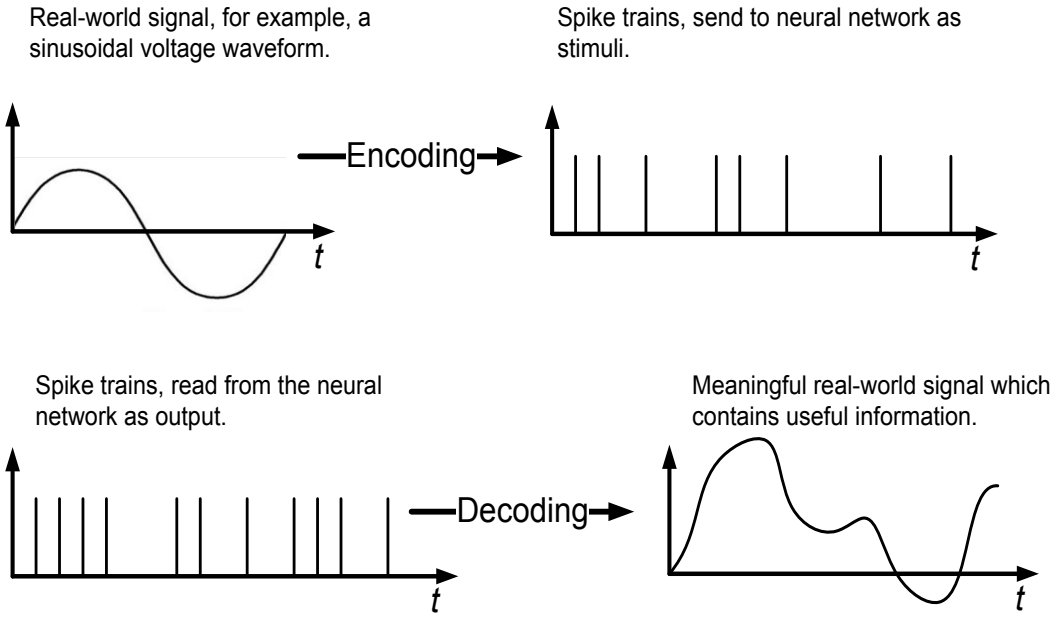


Figure 3.3: Neuronal encoding and decoding.

A sequence of spikes may contain completely different information based on different coding schemes. A number of encoding methods, each with its own advantages and disadvantages have been proposed by researchers over the past 90 years. The most commonly used coding approach is based on the firing rates of individual neurons, but this is by no means the only option. Temporal coding is another commonly used neuronal coding method in which the precise temporal structure of the spike train is taken into account [79-82].

3.2.1 Rate Coding

The rate coding model of neuronal firing communication states that as the intensity of a stimulus increases, the frequency or rate of action potentials, or "spike firing", increases. Rate coding is sometimes called frequency coding. This phenomenon was originally shown [83] in 1926. In this simple experiment different weights were hung from a muscle. As the weight of the stimulus increased, the number of spikes recorded from sensory nerves that control the muscle also increased. From these original experiments Adrian and Zotterman [83] concluded that action potentials were unitary events, and that the frequency of events, and not individual event magnitude, was the basis for most inter-neuronal communication. From these experimental results, Adrian and Zotterman postulated that most, if not all, of the relevant information was contained in the mean firing rate of the neuron.

Various ways of calculating the mean firing rate have been proposed. Different averaging procedures, which are spike count over some time window [84] and spike activities in a population of neurons [85], can be used to calculate the mean firing rate.

Rate coding is a type of temporal averaging method. It can work well in cases where the stimulus is constant or slowly varying and does not require a fast reaction [74]. In rate coding schemes, all spikes are treated equally, in other words, it is the number of spikes over a certain period of time that matters, not the time when the spikes are fired.

3.2.2 Temporal Coding

Some researchers then argued that it is not enough to only consider the firing rate of a neuron, but that spike timing also needed to be taken into account since useful information could be contained in it. The concept of temporal coding, a type of neural coding which relies on precise timing of action potentials or inter-spike intervals was then proposed [86]. Combined with traditional rate coding models, temporal coding can provide additional information with the same rate. There is no precise definition of temporal coding. Actually almost any coding scheme that is not rate coding can be referred to as a temporal coding. Some commonly used temporal coding methods include the time-to-first-spike-based method [87], the coding-by-phase method [88], the correlations and synchrony method [86]. The time-to-first-spike-based method generally thinks that the first spike of a neuron contains most of the information, hence is the most important one; the coding-by-phase methods is often used [86] in situations where the stimulus is not a single event but a periodic signal, which can evoke repeating spike patterns; and the correlations and synchrony methods takes this reasoning one step further, claiming that correlations between spikes within a spike train may carry additional information above and beyond the simple timing of the spikes [86].

3.2.3 Comparison of Rate Codes and Temporal Codes

A formal definition of rate coding was proposed by Theunissen and Miller in 1995 [89] who concluded that if all information contained in a spike train could be recovered by the linear reconstruction procedure of (3.1), then the neuron was using a rate coding approach. Theunissen and Miller assumed that if the set of firing times of a spike train was denoted as $\mathcal{F} = \{t^{(f)}, f = 1, 2, 3, \dots, n\}$, and $s(t)$ was the stimulus signal that needed to be encoded into spikes, and $s^{est}(t)$ was the signal recovered from the encoded spikes, and k was some kind of kernel (or a function) determined through optimization, then the average reconstruction error $\int [s(t) - s^{est}(t)]dt$ would be a minimal, and a spike at time $t^{(f)}$ would give a contribution $k(t - t^{(f)})$ to the estimation $s^{est}(t)$

$$s^{est}(t) = \sum_{f=1}^{n} k(t - t^{(f)}) \tag{3.1}$$

If a linear reconstruction could not be successfully found for a coding method using (3.1), this method is a temporal coding method.

When it comes to choosing proper coding schemes for the BIANN, there is still no definite answer, since which coding method to use is highly dependent on what application the BIANN is designed for. There are two key rules we must follow:

1. Significant information loss during coding is not allowed. When a real-world signal is encoded into a train of spikes, or a spike train coming out of spiking neuron is transformed into a continuous signal, there should always be a reverse algorithm, through which the original signal can be recovered;

2. The coding method should offer a neural network system the possibility to react to the input. Neurons activities have their own time constant. For example, the time for human beings to recognize visual scenes is on the order of a few hundred milliseconds, if

one encodes a stimulus into a spike train with a frequency much higher than a few Hertz, the neuron would not have enough time to process such large amount of data and react correctly.

3.3 Spiking Neuron Models

In any study of network dynamics, there are two crucial issues: (1) what model describes spiking dynamics of each neuron and (2) how the neurons are connected. Inappropriate choice of the spiking model or the connectivity may lead to results unrelated to the information processing by the brain [90]. In this section, some of the mostly used models of spiking and burst neurons are reviewed and compared in terms of their biological plausibility, computational requirement and applicability to large-scale simulations.

Throughout this section, v denotes the membrane potential and v' denotes its derivative with respect to time. All the parameters in the models are chosen so that v has mV scale and the time has ms scale. Also if a model can be written as a dynamic system $\dot{x} = f(x)$, we assume that they can be implemented using fixed-step first-order Euler method $x(t+\tau) = x(t) + \tau f(x(t))$ with the integration time step τ chosen to achieve a reasonable numerical accuracy.

3.3.1 Leaky Integrate-and-Fire (I&F) Neuron

The leaky integrate-and-fire (I&F) neuron model [91] is one of the earliest models in neuroscience. It can be described using (3.2).

$$v' = I + a - bv, \text{ if} \geq v_{thresh} ,$$
$$\text{then } v \leftarrow c \tag{3.2}$$

where v is the membrane potential, I is the input current, and a, b, c, and v_{thresh} are the

parameters. When the membrane potential v reaches the threshold value v_{thresh}, the neuron will fire a spike, and v is reset to c.

The leaky I&F neuron model is the simplest model to implement. When the integration step τ is chosen to be 1 ms, (3.2) can be rewritten into the iterative form (3.3):

$$v(t+1) = v(t) + (I + a - bv(t)) \tag{3.3}$$

The disadvantage of the leaky I&F model is that it is one-dimensional (1-D), in other words, it has only one variable, so it can fire tonic spikes with constant frequency. It cannot have phasic spiking, bursting of any kind, rebound responses, threshold variability, bistability of attractors, or autonomous chaotic dynamic. Moreover, because of the fixed threshold, the spikes do not have latencies.

In summary, despite its simplicity, the leaky I&F model is not a good choice for the BIANN research and simulation since it cannot accurately represent some important neuron features.

3.3.2 Leaky I&F Neuron Model with Adaptation

Since the leaky I&F model can only fire tonic spikes with constant frequency, an improved form of the model shown in (3.4) is developed, which endows it with spike-frequency adaptation [90].

$$\begin{aligned} v' &= I + a - bv + g(d - v) \\ g' &= \frac{(e\delta(t) - g)}{\tau} \end{aligned} \tag{3.4}$$

The activation dynamics of a high-threshold K current is described by the second equation. Each firing increases the K^+ current gate g via Dirac delta function δ and produces an outward current that slows down the frequency of tonic spiking.

The model adds spike frequency adaptation to the leaky I&F model, but lacks many important properties of cortical spiking neurons.

3.3.3 Integrate- and-Fire-or-Burst Model

A further improvement of the leaky I&F model, the integrate-and-fire-or-burst (I&FB) model, has been proposed in [92] to model thalamo-cortical neurons. The dynamics of the model is shown in (3.5).

$$v' = I + a - bv + gH(v - v_h)h(v_T - v)$$

if $v = v_{thresh}$, then $v \leftarrow c$

$$h' = \begin{cases} \frac{-h}{\tau^-}, & \text{if } v > v_h \\ \frac{(1-h)}{\tau^+}, & \text{if } v < v_h \end{cases} \tag{3.5}$$

Here h describes the inactivation of the calcium T-current, g, v_h, v_T, τ^+ and τ^- are parameters describing dynamics of the T-current, and H is the Heaviside step function.

Compared to the leaky I&F adaptation model, the IF&B model creates the possibility for phasic bursting, phasic spiking, rebound spike, rebound burst and bistability. However, the IF&B model has a higher computational cost than the leaky I&F model and the leaky I&F adaptation model.

3.3.4 Resonate-and-Fire Model

The resonate-and-fire neuron model [93], which can be described in (3.6) is a two-dimensional (2-D) analogue of the leaky I&F model.

$$z' = I + (b + i\omega)z \quad \text{if Im(z)} = a_{thresh},$$

$$\text{then } z \leftarrow z_0(z) \tag{3.6}$$

where the real part of the complex variable z is the membrane potential. Here b, ω, and

a_{thresh} are parameters, and $z_0(z)$ is an arbitrary function describing activity-dependent after-spike reset.

The resonate-and-fire model is simple, but still cannot represent some important neuro-computational features.

3.3.5 Quadratic I&F Model

The quadratic I&F neuron, also known as the theta-neuron or the Ermentrout-Kopell canonical model [94, 95] is an alternative to the leaky I&F model. It can be presented as (3.7) following [96].

$$v' = I + a(v - v_{rest})(v - v_{thresh}) \text{ if } v = v_{peak},$$
$$\text{then } v \leftarrow v_{reset} \tag{3.7}$$

where v_{rest} and v_{thresh} are the resting and threshold values of the membrane potential.

This model should be the model of choice when one simulates large-scale networks of integrators since the model is low in computational cost.

3.3.6 Fitzhugh-Nagumo Model

The Fitzhugh-Nagumo Model [97] is shown in (3.8):

$$v' = a + bv + cv^2 + dv^3 - u$$
$$u' = \varepsilon(ev - u) \tag{3.8}$$

The parameters in the model can be tuned so that it can describe spiking dynamics of many resonate neurons. The required simulation time step is relatively smaller than other models mentioned earlier in Section 3.3 since one needs to simulate the shape of each spike using this model.

3.3.7 Hindmarsh-Rose Model

The Hindmarsh-Rose model of the thalamic neuron [98] can be written as (3.9):

$$v' = u - F(v) + I - w$$

$$u' = G(v) - u$$

$$w' = \frac{(H(v)-w)}{\tau} \tag{3.9}$$

where *F*, *G* and *H* are some functions. The model can, in principle, exhibit most of the neuro-computational properties by properly choosing these functions. However, the problem is how to find the functions. For different types of neurons, these functions will be different. So this model is an optimistic assessment that might never be achieved.

3.3.8 Morris-Lecar Model

The Morris-Lecar model [99] is a simple 2-D model to describe oscillations in barnacle giant muscle fiber. It has bigphysically meaningful and measurable parameters, so the model became quite popular in computational neuroscience community.

The model can exhibit various types of spikes, but requires a simulation time step that is significantly smaller than 1 ms, e.g., $\tau = 0.1$ ms is the largest step that gives reasonable results when the model is used to simulate cortical spiking neurons. In additional, the Morris-Lecar model contains hyperbolic tangent (tanh) and exponents, the computational efforts is relatively more than other models.

3.3.9 Wilson Polynomial Model

The Wilson Polynomial Model [100], which contains four differential equations, uses polynomial equations to model cortical neurons. The model can exhibit most of the important neuron computational properties if the parameters are chosen appropriately,

which is difficult. And the suggested simulation time step in the model is 0.1 ms. The number can be pushed up to 0.25 ms without significant loss of precision or noticeable distortion of the shape of the spikes.

3.3.10 Hodgkin-Huxley Model

The Hodgkin-Huxley Mode [101] is one of the most important models in computational neuroscience. It consists of four differential equations and tens of parameters. If the parameters are tuned properly, the model can exhibit most of the neuron properties. The parameters used in the Hodgkin-Huxley model are biophysically meaningful and measurable.

The drawback is that the model is extremely expensive to implement. Thus, it is preferable to only use the model to simulate a small number of neurons or when simulation time is not an issue.

3.3.11 Izhikevich Model

A simple model of spiking neurons proposed recently by Izhikevich [102] is shown in (3.10).

$$\begin{cases} v' = 0.04v^2 + 5v + 140 - u + I \\ u' = a(bv - u) \end{cases} \text{, if } v \geq 30 \text{ mV, then } \begin{cases} v = c \\ u = u + d \end{cases} \tag{3.10}$$

Here variable v represents the membrane potential of the neuron and represents a membrane recovery variable, which accounts for the activation of K^+ ionic currents and inactivation of Na^+ ionic currents and it provides negative feedback to v. After the spike reaches its apex (+30 mV), the membrane voltage and the recovery variable are reset.

The model can exhibit firing patterns of all known types of cortical neurons with the choice of parameters a, b, c and d given in [28]. It takes only 13 floating point operations to simulate 1 ms of the model, so it is quite efficient in large-scale simulations of cortical networks.

3.3.12 Summary of Previously Proposed Neuron Models

In conclusion, a good neuron model should have two important features: computationally efficient and biologically plausible. Computationally efficient means (1) small number of variables, (2) easy form of equations for describing neuron dynamics and (3) low requirement in simulation time step. Biologically plausible means (1) the neuron model should be connected according to the principles of known anatomy of the neocortex and (2) the neuron model should be able to reproduce the rich and complex spiking behaviors of the real neurons.

20 of the most prominent features of biological spiking neurons are reviewed in [90] to illustrate the richness and complexity of spiking behavior of individual neurons in response to simple pulses of dc current, as shown in Figure 3.4 (Each horizontal bar denotes a 20-ms time interval).

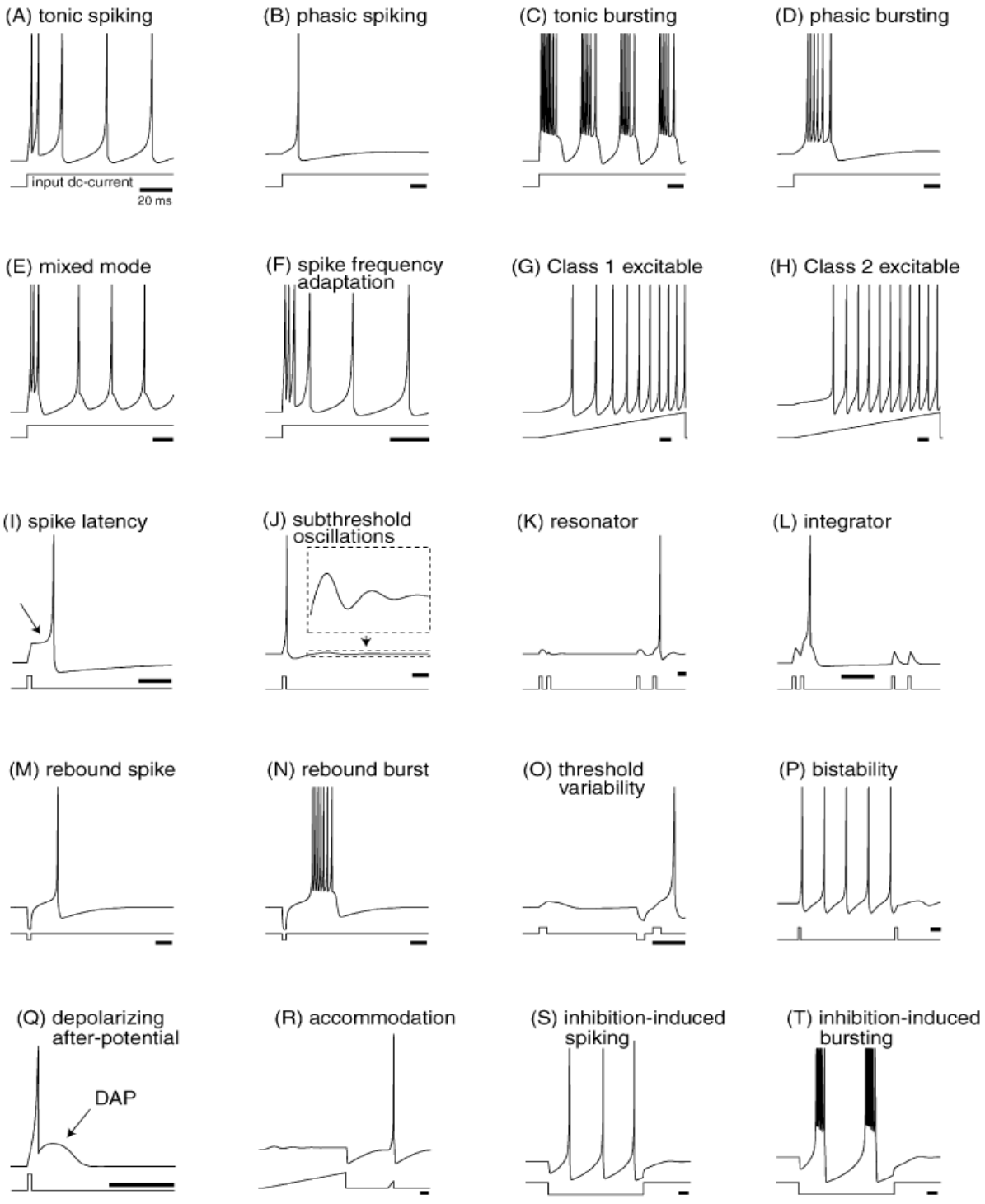
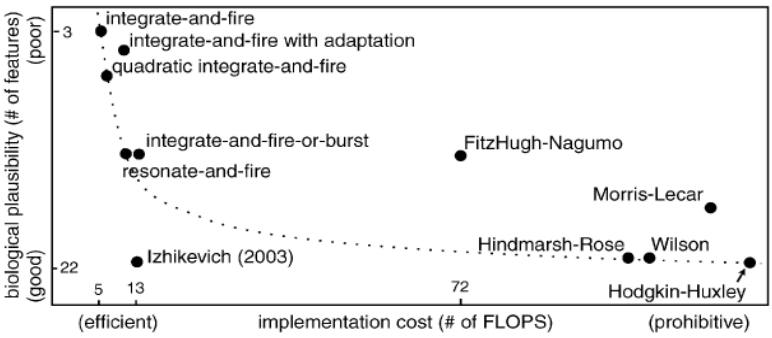


Figure 3.4: Summary of the neuro-computational properties of biological spiking neurons [86].

A comprehensive comparison of the neuron models mentioned in this section has also been given in [90], as shown in Figure 3.5. In Figure 3.5, "# of FLOPS" is an approximate number of floating point operations (addition, multiplication) needed to simulate the model during a 1 ms time span. Each empty square indicates the property that the model should exhibit in principle (in theory) if the parameters are chosen appropriately, but the author of [90] failed to find the parameters within a reasonable period of time. It can be seen from Figure 3.5 that the Izhikevich model can represent all the firing patterns summarized in Figure 3.4 without introducing extensive computational complexity.

3.4 Chapter Summary

To bridge the gap between ANNs and LNNs, a novel type of artificial neural networks, i.e. BIANNs, is highly desired for assisting the study on LNNs and their learning, information processing and storage mechanisms and developing approaches to utilize such huge computational potential in practical applications. A comprehensive literature review has been presented in this chapter on the state-of-the-art biologically inspired spiking neuron models and neuronal coding methods. Various encoding/decoding approaches and various neuron models have been reviewed in detail in terms of their effectiveness, completeness and computational simplicity. The selection of the coding approaches is crucial for the effectiveness of BIANNs in terms of information transmission effectiveness. The selection of the spiking neuron model is also essential for the BIANNs, which is a tradeoff among modeling capability, computational complexity.



Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-induced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figure 3.5: Comparison of neuro-computational properties of spiking and bursting models [86].

CHAPTER 4 ECHO STATE NETWORKS FOR HARMONIC CURRENT IDENTIFICATION

4.1 Overview

With the wide use of power electronics devices, large amounts of harmonic currents are injected into the power system, a phenomenon known as “harmonic pollution”. The “harmonic pollution” jeopardizes the power quality of the power system and might cause damage to certain loads that have low harmonic tolerance levels, so the IEEE standard 519 and the IEC-1000-3 have been established to limit the amount of harmonic current and voltage generated by utilities and customers. Accurately measuring how much harmonic current is being injected into the power system and then eliminating it with proper devices are important to the power system.

When the voltage at the point of common coupling (PCC) is sinusoidal, the harmonic current generated by a nonlinear load can be measured and processed directly from the load current and these harmonics are defined as the “true harmonic contributions” from this load. However, when the supply voltage itself is distorted, due to other nonlinear loads and saturating transformers further upstream in the power network, the load current contains harmonics caused both by the supply (referred to as supply harmonics) as well as the nonlinear load (referred to as load harmonics). Simply now measuring the current waveforms of such a load, yields the combination of load harmonics and supply harmonics, and do not yield the true distortion (load harmonics only) caused by the load. Due to these complicated interactions between loads and sources, it is a challenge to identify the true harmonic current generated by an individual nonlinear load.

One method to address this harmonic issue as proposed by Mazumdar [103], was to use an advanced neural-network-based harmonic current detection scheme to first estimate the true harmonic current contributed by the nonlinearity of the load, instead of by the distorted power supply. This approach can separate out the contributions of harmonic current distortion caused by the system and caused the end-user. Then, an active power filter can be used to compensate selected harmonic current [104-111], leaving the source current flowing out of the PCC clean and nearly purely sinusoidal. This neural-network-based harmonic detection and compensation scheme is shown in Figure 4.1. The nonlinear load modeling part is demonstrated in this chapter and the active harmonic filtering part is discussed in Chapter 5.

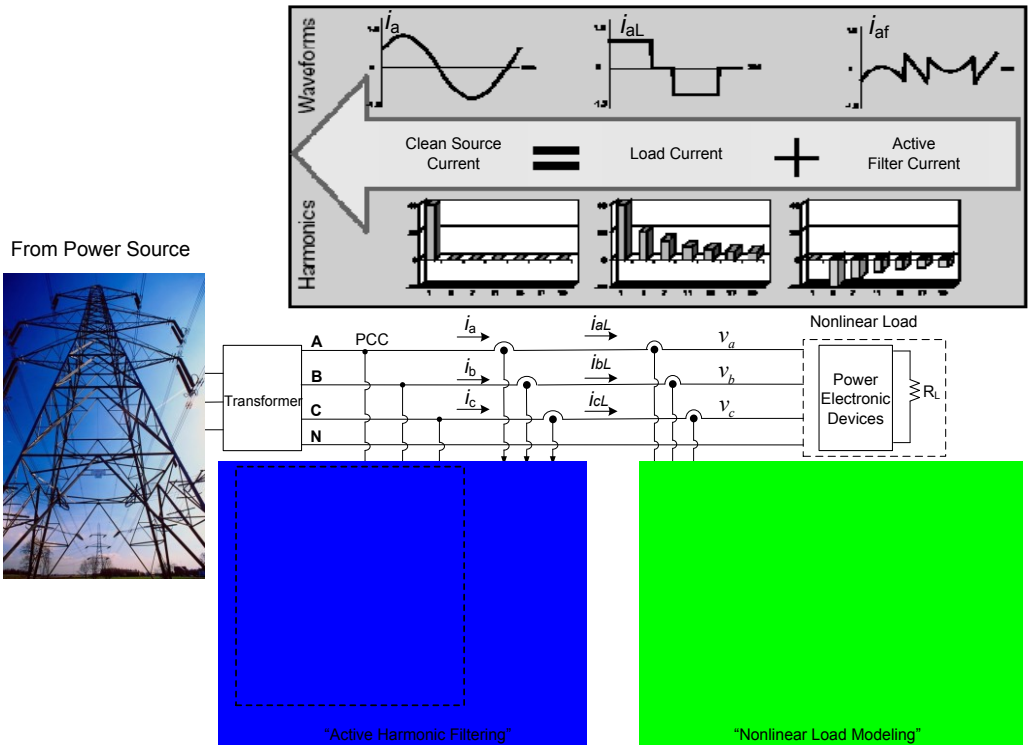


Figure 4.1: Neural-network-based harmonic detection and compensation scheme.

Since a power system is a large-scale, nonlinear, non-stationary system with varying dynamic characteristics over a wide range of operating conditions, the active power filter

and the associated power network cannot be accurately modeled as a linear system with fixed and known parameters. Therefore, when the operating conditions of the system change or in the case of a major disturbance, the performance of a linear controller of the active filter that is designed for a certain operating point degrades or may even become unstable. According to the literature review in Chapter 2, ANNs are good at identifying nonlinear systems and avoiding the need for an explicit mathematical model of the power system; such an ANN identifier can then be combined with a nonlinear intelligent ANN controller, to provide effective control for the system over a wide range of operating conditions. The question then arises: which type of neural network architecture to use as the ANN identifier and the controller? The echo state network (ESN) is chosen for its fast training speed.

4.2 Echo State Networks

The ESN is a special type of recurrent neural network. To maintain the good modeling capability of ordinary recurrent neural networks, a large (e.g. 100 hidden neurons) RNN is used as a "Dynamic Reservoir" (DR) in the hidden layer of the ESN, which can be excited by suitably presented input and/or feedback of the output. The architecture of the ESN is shown in Figure 4.2. Due to the complex structure of the DR, the ESNs are good at modeling complex dynamic systems. However, to limit the required computational effort during training, novel types of offline and online training algorithms have been proposed by Jaeger [11, 67] and Venayagamoorthy [72] respectively, which are summarized in Sections 4.2.1 and 4.2.2.

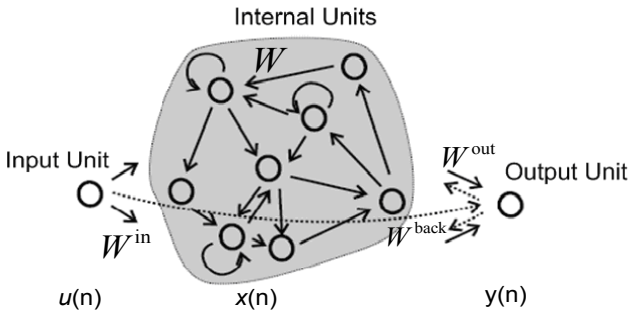


Figure 4.2: The structure of the ESN.

4.2.1 Offline Training Algorithm of the ESN

The steps of the offline training algorithm for the ESN are summarized below:

1. Generate a recurrent neural network following certain rules to ensure its "echo state property [67]".

Three weight matrices should be generated. They are the input weight matrix W^{in}, internal weight matrix W and the output feedback matrix W^{back}. Once W^{in}, W and W^{back} have been generated, they will not change during the entire training process. The echo state property is related to the algebraic properties of the weight matrix W; however, there is no known necessary and sufficient algebraic condition which indicates whether the network has the echo state property, given W^{in}, W and W^{back}. Nevertheless, there are certain conditions which increase the possibility of the RNN to have the echo state property. Usually W is generated by following the principles described below:

2. Generate a sparse matrix W_0 and make sure the mean value of all the weights in it is about zero.

Normalize W_0 to a matrix W_1 with unit spectral radius following (4.1):

$$W_1 = \frac{W_0}{|\lambda_{max}|} \quad , \tag{4.1}$$

where λ_{max} is the spectral radius of W_0 calculated as follows:

Scale W_1 to W, using (4.2):

$$W = \alpha W_1 \quad (\alpha < 1) \tag{4.2}$$

3. Feed the training data to the ESN.

Feeding the training data into the ESN will activate the dynamics within the dynamic reservoir. At each sampling step, compute the internal dynamic reservoir states according to (4.3):

$$x(n+1)=tanh(u(n+1)W^{in}+x(n)W+y(n)W^{back}) \tag{4.3}$$

where u is the input vector, x is the vector of internal units and y is the output vector.

4. Collect the states at time n as a new row in a matrix DR, which is a matrix used to store the dynamic reservoir values during the training.

Since at sampling step $n = 0$, $x(0)$ and $y(0)$ are not defined, the following initial conditions are used:

(i) Initialize the network state arbitrarily, e.g. to the zero state $x(0)=0$;

(ii) Set $y(0)=0$.

5. Wash out the initial memory in the DR.

Since the arbitrarily generated network states (e.g. zero state) contain an initial memory which is not caused by the input, do not collect information from times $n = 1, 2, \ldots, n_0$. (n_0 is different according to different systems and the length of input sequence.)

By time $n = n_0 + 1$, it is safe to assume that the effects of the arbitrary starting state have died out and that the network states are a pure reflection of the teacher-forced input and output.

So the first n_0 rows of the states matrix DR are removed to obtain a new matrix DR^{forget}.

6. Compute the output weights.

The first n_0 rows of the teacher output sequence $y(n)$ are also removed to obtain a new matrix $Teacher^{forget}$.

Then the output weight is calculated using (4.4):

$$W^{out} = (Pseudoinverse(DR^{forget}) \cdot Teacher^{forget})^T \quad (4.4)$$

4.2.2 Online Training Algorithm of the ESN

The online training algorithm of the ESN proposed in [72] is described as follows:

1. Generate a recurrent neural network

Different from the offline training algorithm, there are **four** initial weight matrices that should be generated. In addition to the input weights W^{in}, the internal weights W and the feedback weights W^{back}, and the output weights W^{out} are all randomly generated at the beginning.

The rules for generating the internal weights W are the same as those described in the offline training algorithm in Section 4.2.1.

2. Calculate the states in the Dynamic Reservoir

This step is the same as described in the offline training algorithm. (4.3) is used for the calculation of the states in the Dynamic Reservoir. However, it is no longer required to collect the states at time n as a new row of a state matrix DR.

3. Compute the estimated output of the ESN

Now the output $\hat{y}(n)$ of the ESN is calculated by (4.5):

$$y(n+1) = x(n+1)W^{out}(n)\ , \tag{4.5}$$

where $W^{out}(n)$ denotes the output weight matrix at time step *n*.

4. Update the output weights

The estimated output $\hat{y}(n+1)$ is compared with the actual output $y(n+1)$ collected from the actual system and the error vector e_y is calculated as (4.6):

$$e_y(n+1) = y(n+1) - \hat{y}(n+1)\ , \tag{4.6}$$

The output weights are updated according to (4.7), as described in:

$$W^{out}(n+1) = W^{out}(n) + \eta x^T(n+1)e_y(n+1) + \gamma x^T(n)e_y(n), \tag{4.7}$$

where η is the learning gain and γ is the momentum gain, and each one is in the range of [0, 1], similar to the parameters used in other types of neural network training schemes.

The output weights are updated such that the mean squared training error (MSE) is minimized according to (4.8):

$$MSE = \frac{1}{r}\sum_{n=1}^{r}(y(n) - \hat{y}(n))^2, \tag{4.8}$$

where r is the length of the Input/Output sequence used for training.

In both online and offline training algorithms of ESN, only the output weights are updated, which guarantees the low computational requirement of ESN.

4.3 Harmonic Current Identification for Power System Nonlinear Loads

A novel load modeling approach, which is based on neural networks to identify the true harmonic contribution, has already been proposed and validated by laboratory experimental data [103, 112]. Since neural networks are known as effective tools of function approximation, the relationship between the input voltage and output current of a load can be learned by a neural network after proper training; and then, if fed with another input voltage, the output current of the neural network should be the same as the

current of the load. Therefore, the properly trained neural network can serve as an accurate model or identifier of the nonlinear load to predict the true distorted current attributed to the load. The schematic diagram of the load modeling approach is shown in Figure 4.3.

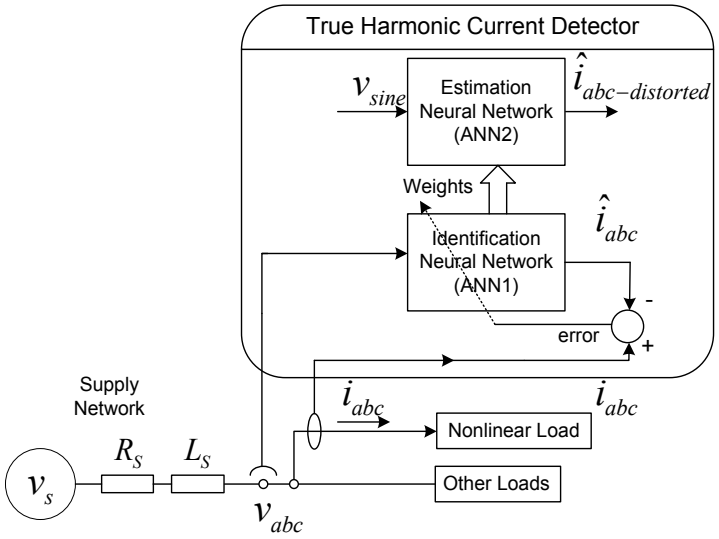


Figure 4.3: ANN-based load harmonic current identification and estimation system.[99]

Reference [99] described that the distorted line current i_{abc} and the distorted power supply voltage v_{abc} could be measured to identify the nonlinear characteristic of the load. The identification neural network (ANN1) was then trained to learn the nonlinear characteristics. The estimation neural network (ANN2), which is an exact replica of the trained ANN1, was used to predict the true current harmonics.

The identification neural network (ANN1) was trained with the measured distorted voltage v_{abc} as input and the measured distorted load current i_{abc} as output. After training, a mathematically generated sine wave was supplied to the estimation neural network ANN2, which is a replica of ANN1, with the latest trained weights of ANN1, to predict

the load current harmonics when the load was supplied by a "clean" sinusoidal voltage source which contained very little or no harmonics. Ideally, ANN1 learned the exact nonlinear characteristics of the load, and therefore the distortion present in the output of ANN2 could be considered as the exact or true load current and its harmonics attributed to the nonlinearity of the load only.

Due to the nature of the sigmoid transfer function used in the MLPs in this application, the inputs to both ANN1 and ANN2 were scaled to [-1, 1].

4.4 A Comparison of ESNs, MLPs and RNNs in Harmonic Current Identification

4.4.1 Experimental Setup

The proposed load harmonic current identification scheme was validated by experiments using a drive-connected induction motor as the nonlinear load. The experimental setup used by Mazumdar [113] is shown in Figure 4.4. The variable speed drive (ABB ACS 500) as load was supplied from the utility source which contained background distortion. For validation purposes, a California Instruments 5001 iX harmonic generator, which provided voltages with programmable distortion levels and zero internal impedance, was used as a "clean" power source in order to measure the true current harmonics. The three-phase voltages and currents are measured using LEM LV 25-P voltage transducers and LEM LAH 25-NP current transducers, respectively. The measured data was then acquired and stored using a National Instruments data acquisition system with an Analog-to-Digital resolution of 16 bits, at a sampling frequency of 8 kHz. The scheme was applied to each phase individually. Phase A was used as an example to demonstrate the proposed method.

The measured phase A voltage and current with distorted and clean power supplies, are shown in Figures 4.5 (a) (b) and Figures 4.5 (c) (d), respectively [109].

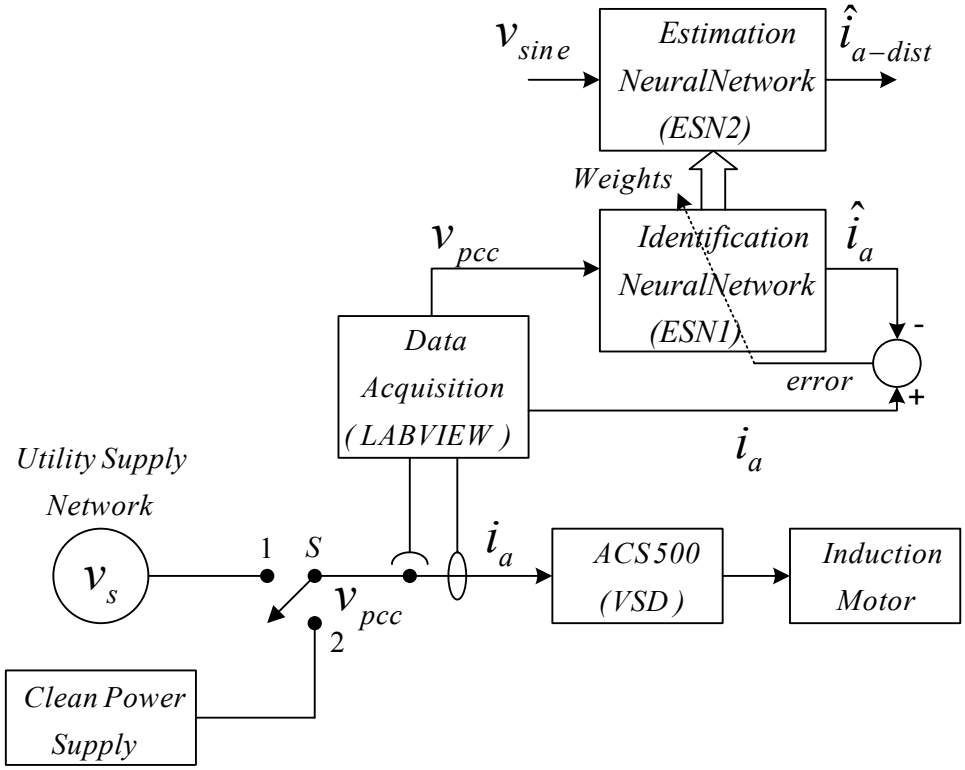


Figure 4.4: Experimental setup for the load harmonic current identification.[109]

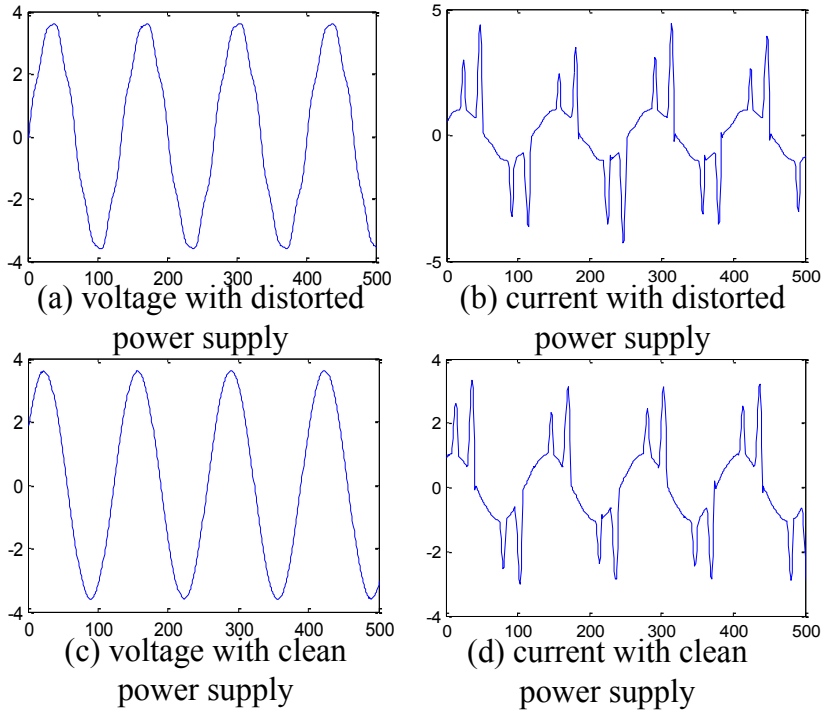


Figure 4.5: Measured voltage and current with distorted and clean power supplies.[109] as a function of time in milli-seconds

The background harmonic distortion of the utility voltage v_{pcc} at position 1 in Figure 4.4 was 4.5%, and the background harmonic distortion of the clean power supply was 0.2%, which were respectively considered as a distorted power supply and a "clean" power supply. The use of the "clean" power supply was only for validation purposes of the algorithm.

The feasibility of using three different types of neural networks: MLP, RNN and ESN with different numbers of hidden neurons in this application were tested, and the performances were compared. The training, testing and prediction results of a MLP and a

RNN are presented in this section. For both networks, the number of neuron in the hidden layer was 15.

4.4.2 Training, Testing and Prediction Data Construction

The voltage and current data measured from the induction motor load shown in Figure 4.4 are separated into 3 parts: training data, testing data, and prediction data. The training data are the first 10 k samples of the measured voltage and current (Figures 4.5(a) and (b)) with the distorted power supply. For the ESN, the first 1 k samples are not used in training, which is the so called "forget" period [67]. The testing data are 6 k samples after the training data, with the distorted power supply, to test the training results. Then 3 k samples of measured voltage and current with the clean power supply are used to test the harmonic current prediction results of the three neural networks

For the ESN, the learning of the load characteristics does not only lie in the trained weights, but also lies in the hidden neuron values, i.e. the states in the Dynamic Reservoir, which can be considered as the "memory" of recurrent neural networks. Therefore the prediction data and the testing data are concatenated for harmonic current prediction, as shown in Figure 4.6.

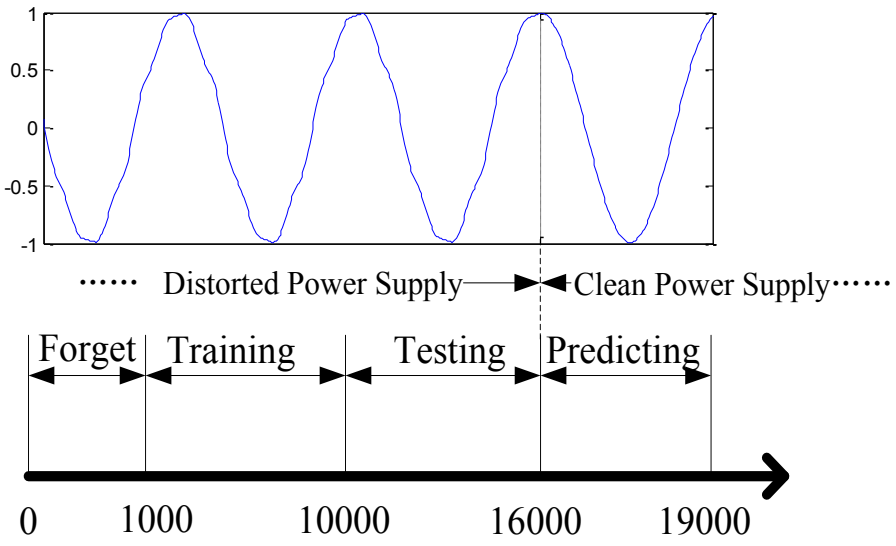
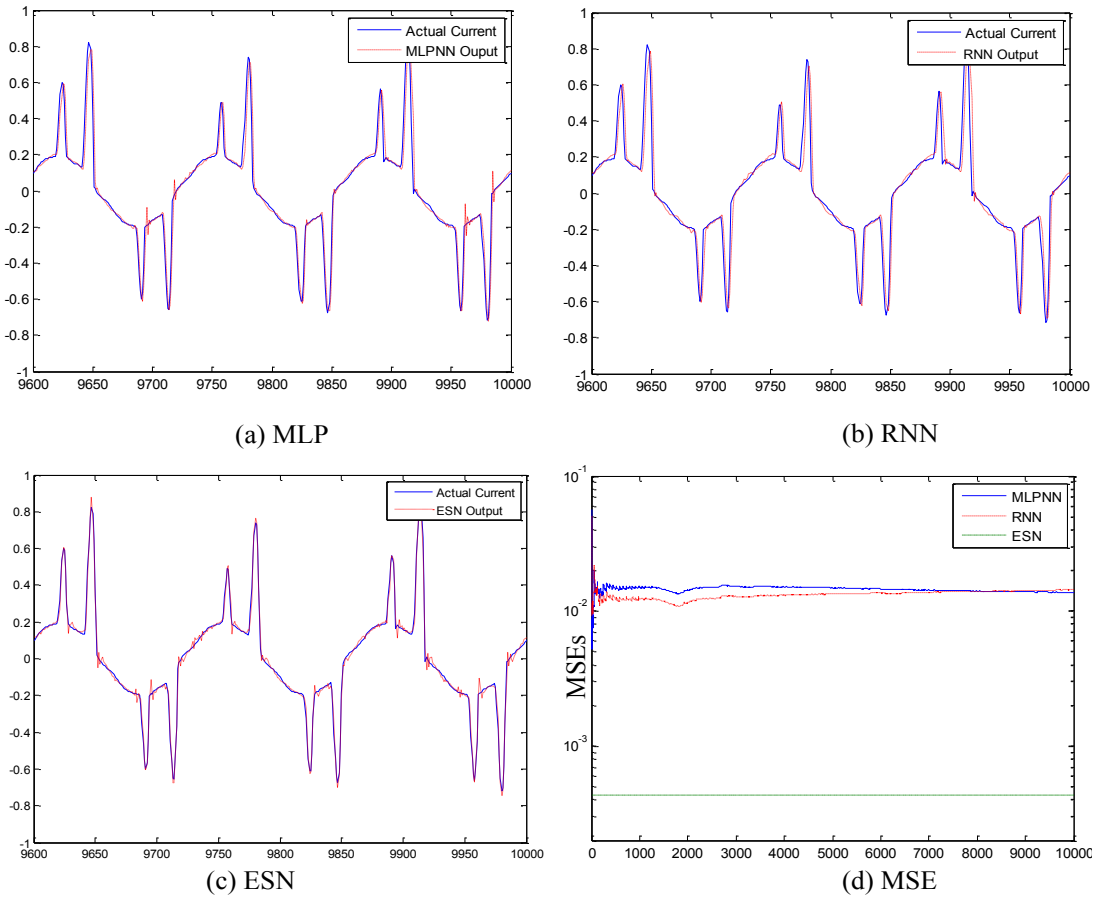


Figure 4.6: Structure of training, testing and prediction data for the ESN.

4.4.3 Neural Network Training Results

The training results of the MLP, RNN and ESN are shown in Figure 4.7(a), (b), and (c), respectively and the Mean Squared Errors (MSEs) of the three neural networks are shown in Figure 4.7(d). In Figs. 4(a) to (c), the Y-axis represents current in per unit, and the X-axis represents time in milli-seconds. The training of the ESN is only a one-step calculation of output weights by pseudo inverse described in Section 4.2.1[11], so there is no step-by-step MSE available.

Figure 4.7(d) shows that the MSEs of the MLP and RNN converge to a similar value, while the MSE of the ESN is much smaller. This is understandable as the training of the ESN can be considered optimal, as the pseudo inverse is used.



(a) MLP (b) RNN

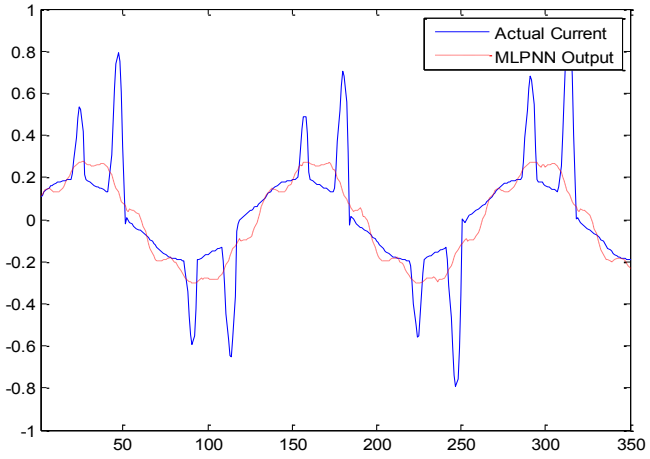
(c) ESN (d) MSE

Figure 4.7: Neural network training results of the MLP, RNN, and ESN as functions of time in milli-seconds.

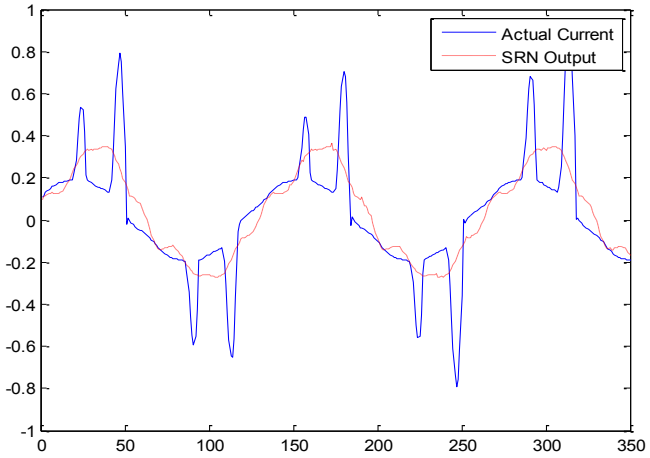
4.4.4 Neural Network Testing Results

To test the learning capability of the three neural networks, a set of voltage and current data with the distorted power supply is fed to the neural networks, and the neural network outputs are compared with the actual output. The testing results are shown in Figure 4.8, which compares the responses of neural network and the actual current for the MLP, RNN and ESN respectively.

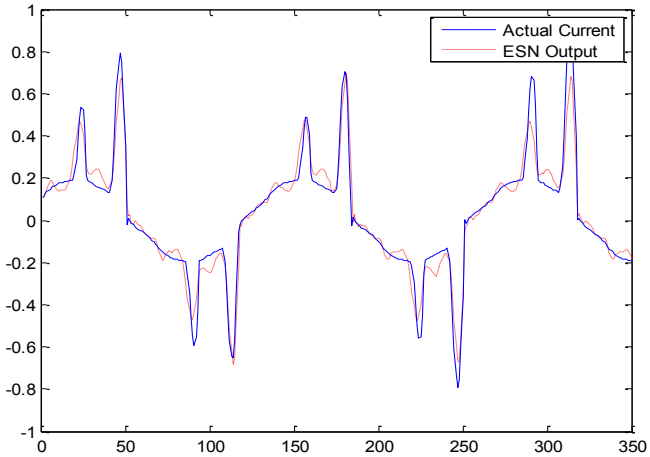
Figure 4.8 shows that the RNN performs slightly better than the MLP, while the ESN gives a much better testing result. From the testing results of the MLP and RNN, it is concluded that the training of the MLP and RNN is insufficient; while the ESN is already sufficiently well trained with the same training data. However, the performance of the ESN is highly dependent on the randomly pre-determined weights of the hidden neurons, which reflect the dynamic features of each hidden neuron in the dynamic reservoir. If the randomly selected weights can accurately represent all the dynamic characteristics of the system, then the ESN can perform accurately as a system simulator.



(a) MLP



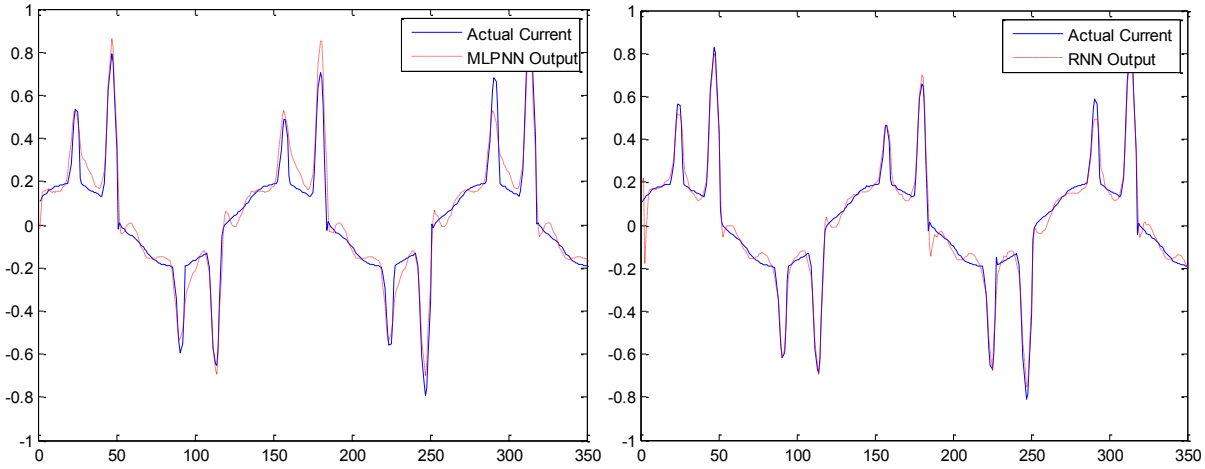
(b) RNN



(c) ESN

Figure 4.8: Neural network testing results of the MLP, RNN, and ESN as functions of time in milli-seconds.

The testing results of sufficiently trained MLPs and RNNs are shown in Figure 4.9. For sufficient training, training data with 500 k samples, instead of 10 k samples, is used. Figure 4.9 shows that the RNN performs better than the MLP network, which is caused by its "recurrent" feature: the outputs are both determined by the inputs and the previous states of the network. This "recurrent" feature makes the recurrent neural network a better tool for dynamic system modeling.



(a) MLP (b) RNN

Figure 4.9: Testing results of the MLP and the RNN after sufficient training as functions of time in milli-seconds.

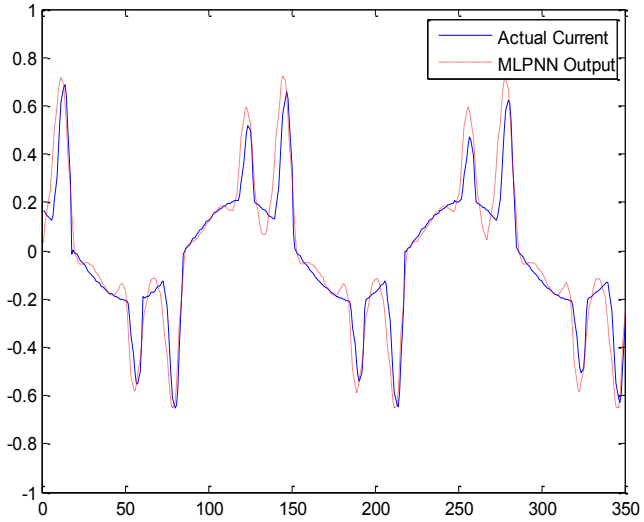
.

4.4.5 Harmonic Current Prediction Results

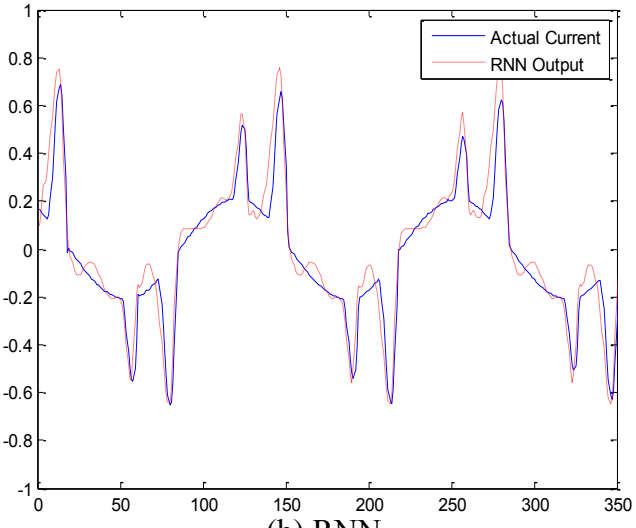
After testing the learning capability of the neural networks, a pure sinusoidal wave is supplied to the trained neural networks to predict the actual distorted current attributed to the nonlinearity of the load.

Figures 4.11(a), (b), (c) show the currents predicted by the MLP, RNN and ESN, respectively, as well as the actually measured currents with the "clean" power supply.

The predicted results of sufficiently trained MLP and RNN networks are shown in Figure 4.10. Comparing Figure 4.10 and Figure 4.11 shows that, like the testing results, the performances of the RNN and ESN in harmonic current prediction are better than the MLP, which is caused by their superior capability of dynamic system modeling. Again, like the testing results, the performance of the ESN is highly dependent on the randomly selected weights of the hidden neurons.

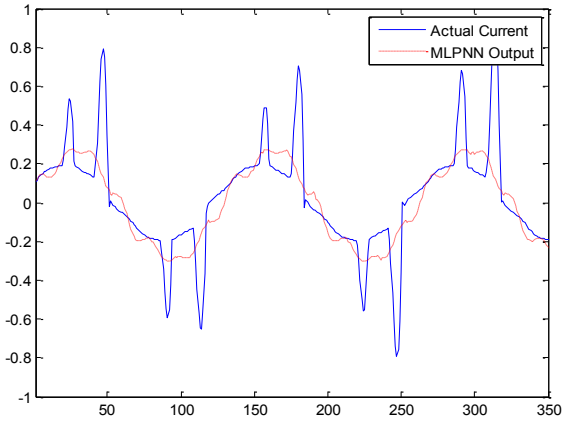


(a) MLP

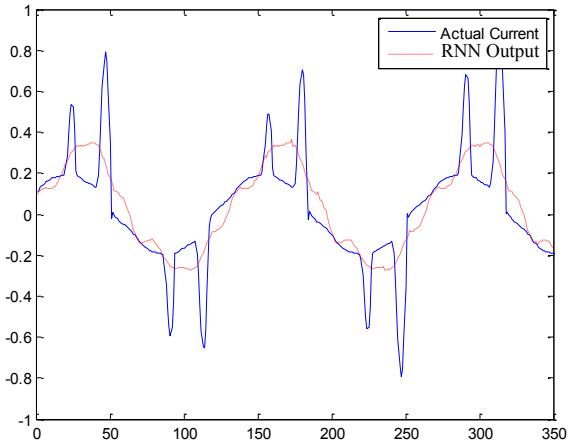


(b) RNN

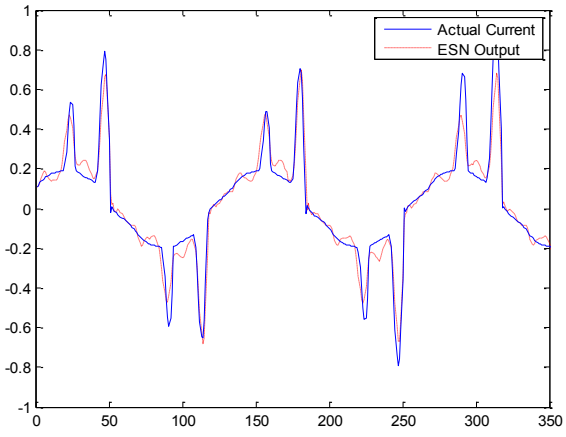
Figure 4.10: Distorted current prediction of the MLP and RNN after sufficient training as functions of time in milli-seconds..



(a) MLP



(b) RNN



(c) ESN

Figure 4.11: Distorted current prediction results of the MLP, RNN, and ESN as functions of time in milli-seconds..

4.4.6 Comparison of the MLP, RNN and ESN

1. Calculation effort comparison

The MSEs of the MLP, RNN and ESN, first with 15 and then with 30 hidden neurons in the hidden layer, in training, testing and prediction are compared in Table 4.1. In this table, the MLP and RNN are both trained with sufficient training data; while a much shorter training data set is used for the training of the ESN. During training, the MSE of the ESN is much smaller than the MSEs of the MLP and the RNN, which are quite close. This is because of the fundamental difference between the training algorithms. In testing and prediction, the MSE of the MLP is larger than the MSEs of the RNN and the ESN, which are still quite close. This implies that the learning capabilities of the ESN and the RNN are better than the MLP in nonlinear load modeling. However, the performance of the ESN is more dependent on the randomly generated initial weights than the MLP and the RNN, as some of the weights in the ESN are not updated during training.

Table 4.1: MSE comparison for training, testing and prediction.

Number of Hidden Neurons	15			30		
Types of Neural Networks	MLP	RNN	ESN	MLP	RNN	ESN
Training MSE_{min}	0.0086	0.0081	5.72e-4	0.0110	0.0098	2.53e-4
Testing MSE	0.0071	0.0042	0.0036	0.0075	0.0053	0.015
Prediction MSE	0.0128	0.0072	0.0049	0.0150	0.0195	0.0095

For the MLP and the RNN, the 15-neuron network and 30-neuron network have similar performances, which indicate that the increase of hidden neurons does not improve the learning capability with a certain length of training data. The performance of the 15-neuron ESN is even better than a 30-neuron ESN. It can be concluded that the 15-neuron ESN is already able to learn the complexity of the system, so an increase of hidden neurons does not imply a better performance. Besides, the increase of hidden neurons in the ESN may greatly change the dynamic characteristics of the dynamic reservoir, which may not enhance the learning capability.

2. Calculation time comparison

The computational time of each neural network is compared in Table 4.2. A PC with Pentium 4 CPU working at 3.4 GHz is used for testing. In training, the computation time of the ESN is shorter than for the MLP and the RNN, which is again the result of different training algorithms. For the ESN, the training is simply some matrices calculated without iteration, while for the MLP and the RNN, the back-propagation algorithm requires more computational effort. In contrast, during testing and prediction, the computational effort required by the ESN is larger than for the MLP and the RNN, because the hidden neurons are sparsely connected, and more computation is required to update the states inside the dynamic reservoir of the ESN for every iteration step. Because of the same reason, an increase in the number of neurons in the dynamic reservoir of the ESN, significantly increases the computational time.

3. Convergence property comparison

The trainings of the MLP and the RNN converge faster than for the ESN, due to the dependence of the ESN on the initial weights. For the MLP and the RNN, the back-

propagation algorithm can provide reliable convergence, although more training data are needed. However, for the ESN, when the initially generated weights cannot represent the dynamics of the system, although the pseudoinverse calculation in training is still giving an accurate training result, during the testing, the ESN can become unstable, as the error increases after several iterations.

Table 4.2: Comparison of the computation time for training, testing and prediction.

Number of Hidden Neurons	15			30		
Types of Neural Networks	MLP	RNN	ESN	MLP	RNN	ESN
Training Time (s)	1.80	1.94	0.64	2.06	2.27	0.73
Testing Time (s)	0.42	0.49	0.77	0.50	0.59	2.02
Prediction Time (s)	1.40	1.80	3.52	1.58	1.94	7.28

4.5 Chapter Summary

An ESN-based harmonic current prediction method, which can estimate the true current harmonics attributed to the nonlinearity of a load, has been validated by experimental results using a variable speed drive-connected induction motor as the nonlinear load.

A performance comparison of three types of neural networks: MLP, RNN and ESN with different numbers of hidden neurons has been shown in detail. The training results,

testing results and harmonic current prediction results have been compared. The required computational effort of each neural network has also been discussed. It has been shown that, the MLP and the RNN require a much larger size of training set than the ESN to achieve the same accuracy. The RNN and the ESN produce more accurate load modeling results than the MLP, but require more computational effort. The choice between the RNN and the ESN is a tradeoff between convergence property and the size of the necessary training data. In other words, when the training data is not sufficient, which is quite possible in practice, the ESN can give better system approximation results than the MLP and the RNN.

The ESN is an advanced type of ANN, which is computationally powerful with strong modeling capability. It has the potential to be used in real-time modeling and control applications in the power system.

With the true harmonic distortion attributed to the nonlinear load estimated, it is critical to compensate for the harmonic distortion caused by this load. Due to the limitations of linear controllers, more "intelligent" control of the active power filters is desired to more reliably improve the power quality in the power systems.

CHAPTER 5 ECHO STATE NETWORKS FOR ACTIVE POWER FILTER CONTROL

5.1 Overview

It has been shown in Chapter 4 that the ESN has excellent performances in power system nonlinear load true harmonic calculations. This is only the first step to address the power system harmonic pollution problems. No matter who causes the harmonic distortion, it is critical to remove these harmonic distortions to improve the power quality of the system. Active power filters (APFs) are effective tools to filter the harmonic currents injected into the power network. Figure 5.1 shows the structure of an APF connected to a typical power distribution system. The APF is connected to the point of common coupling (PCC) via a three phase inductor L_f. Based on monitoring the harmonics in the three-phase load currents, i_{aL}, i_{bL}, i_{cL}, the APF injects three-phase currents, i_{aF}, i_{bF}, i_{cF}, with the exact harmonics and phase angles to cancel those harmonics present in i_{aL}, i_{bL}, i_{cL}. This is done by controlling the PWM inverter in an appropriate way.

A typical power system is a large-scale nonlinear, non-stationary system with varying dynamic characteristics over a wide range of operating conditions. Traditional linear controllers are designed from a linearized system model with fixed parameters at a specific operating point. However, in a real power system, the APF and the associated power network cannot be accurately modeled as a linear system with fixed and known parameters. Therefore, at other operating points or in the case of a major disturbance, a linear controller's performance degrades and may even become unstable. ANNs are good at identifying nonlinear systems and avoid the need for an explicit mathematical model of the power system; such an identifier ANN can then be combined with a nonlinear

intelligent neurocontroller, to provide effective control for the APF over a wide range of operating conditions. The question then arises: which type of neural network architecture to use for the identifier?

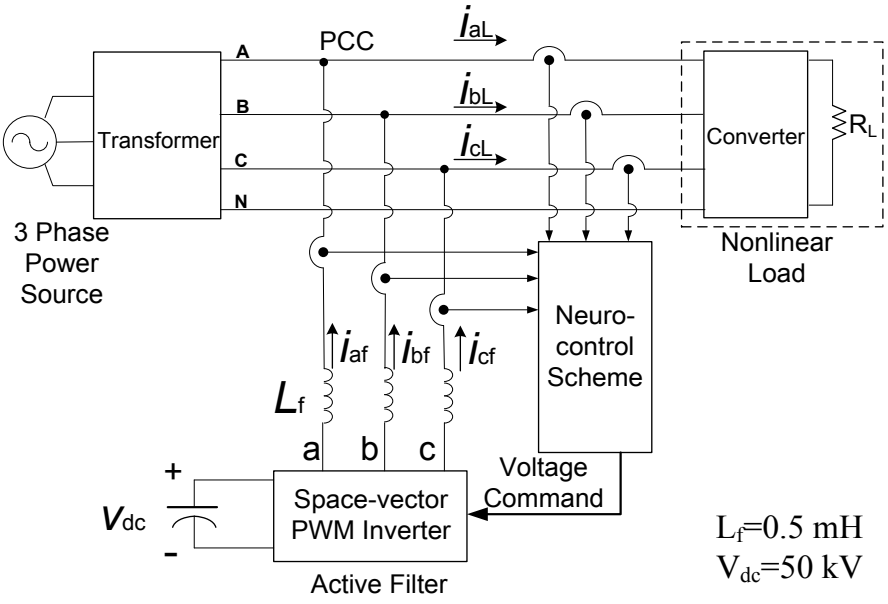


Figure 5.1: An APF connected to a typical power system.

It has been shown in Chapter 4 that the ESN has a faster training speed than the MLP and the RNN. Because of its low training complexity, the ESN has already been used for system identification purposes in various applications [32, 35, 44, 46, 71, 72, 113]. However, few attempts have been made to exploit the feasibility of using an ESN as a system closed-loop controller. As a first attempt to apply ESNs in a closed-loop control system of an APF, this Chapter proposes an indirect adaptive neurocontrol scheme using ESNs.

5.2 Indirect Adaptive Control of the APF using ESNs

Active power filters can be controlled based on various control strategies to obtain current harmonic components for compensation, working either in the frequency domain or in the time domain. In the frequency-domain, a commonly used strategy is to transform the harmonic current components into d,q reference frames, rotating synchronously with the relevant harmonic component, and at multiples of the fundamental frequency, thus transforming each of the respective harmonic components into DC quantities and subjecting them to controllers [114, 115].

The overall scheme for the adaptive neurocontrol of the APF in multiple-reference frames is shown in Figure 5.2.

An AC-DC power electronic converter supplying an adjustable resistance is used as a nonlinear load, which draws harmonic components in the load currents i_{aL}, i_{bL}, i_{cL}. The converter contains thyristors of which the firing angle is adjusted to control the operation of the converter. The 5^{th} and 7^{th} harmonics in the load current, which are the major current harmonics present, are extracted using the method of multiple-reference frames, in this case one reference frame for the 5^{th} harmonic and another for the 7^{th} harmonic. Each reference frame contains an *abc*-to-*dq* transformation but uses an appropriate transformation angle, which rotates at a specific multiple of the fundamental frequency. For example, the 5^{th} harmonic reference frame (HRF) converts only the 5^{th} harmonic components in i_{aL}, i_{bL}, i_{cL} to dc current components i_{d5}^*, i_{q5}^* . A low pass filter then extracts these dc currents and eliminates all the higher frequency components. The i_{af}, i_{bf}, i_{cf} currents are similarly transformed into the 5^{th} HRF and then filtered, and their d, q components i_{d5}, i_{q5} are each compared with the i_{d5}^*, i_{q5}^* respectively, to form the two

errors e_{d5}, e_{q5}. The 7th harmonic currents are processed in a similar way using a 7th HRF and filters to form e_{d7}, e_{q7}. The IEEE 1459 standard provides the definition of harmonic components. The multiple reference frame-based method is adopted to assist fast and reliable control of the system. The e_{d5}, e_{q5}, e_{d7}, e_{q7} are used by the neurocontrol scheme and fed to the neurocontroller to provide the voltage command to the PWM inverter. Space-vector PWM is used to control the power switches in the APF based on the given voltage command. When the errors are minimized by the neurocontroller, the APF injects the 5th and 7th harmonic currents to cancel the harmonic currents caused by the nonlinear load, and hence no 5th and 7th harmonic currents are injected into the power network.

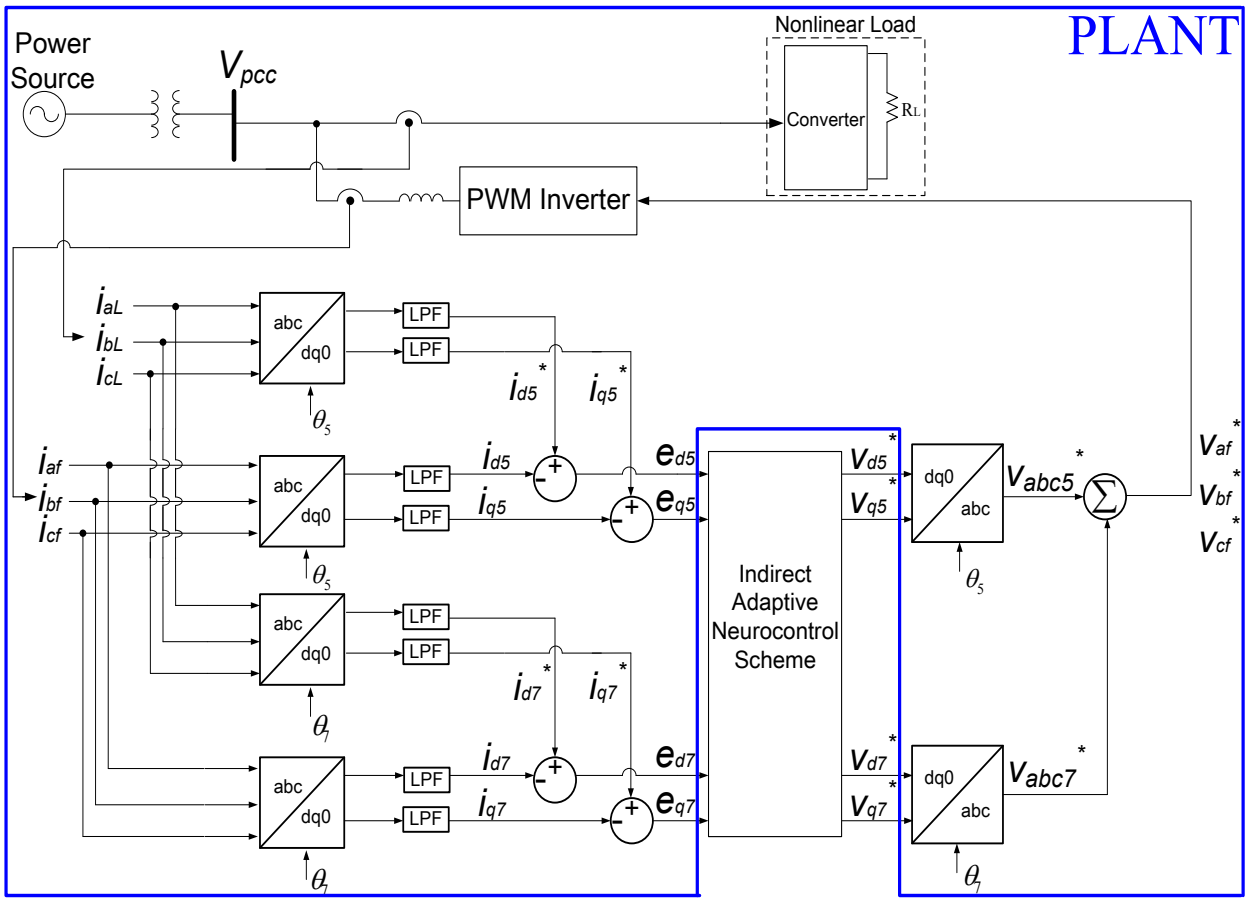


Figure 5.2: The overall multiple-reference-frame-based control scheme of the APF.

The structure of the proposed indirect adaptive ESN-based control appears in Figure 5.3. It consists of two separate ESNs, namely, one as the neuroidentifier and the other as the neurocontroller.

The ESN identifier is used to provide a dynamic model of the PLANT in Figure 5.2 in an online fashion, which predicts the remaining current harmonics in the system given a system input. The Plant input v= [v_{d5}*, v_{q5}*, v_{d7}*, v_{q7}*] and output e_i= [e_{d5}, e_{q5}, e_{d7}, e_{q7}] at time k are fed into the ESN identifier to estimate the PLANT output $\hat{e}_i = [\hat{e}_{d5}, \hat{e}_{q5}, \hat{e}_{d7}, \hat{e}_{q7}]$ at time k+1. The error e_e between e_i and $\hat{e}_i$ (as defined in Figure 5.3) is used to update the

weights inside the ESN identifier. At each time step, the ESN based neurocontroller generates the control signals as the Plant inputs in order to drive the Plant output to the desired value, which is e_i^*=[0, 0, 0, 0], based on the prediction given by the ESN identifier. The error between $\hat{e}_i$ and e_i^* is backpropagated through the ESN identifier to update the weights inside the ESN controller so that the system input *v* can be adjusted to the desired value to minimize error.

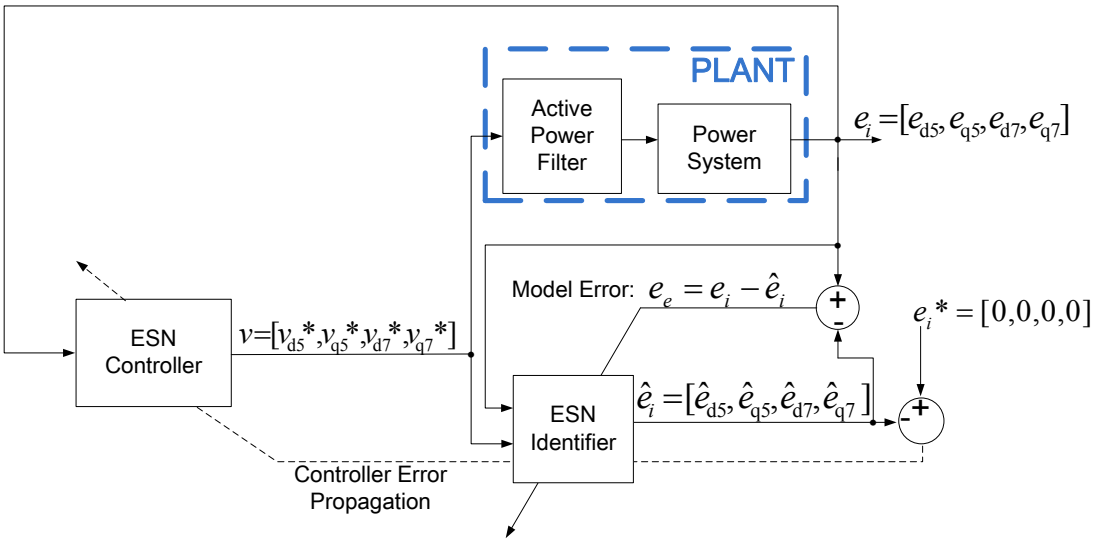


Figure 5.3: Indirect adaptive ESN control scheme using ESNs.

5.3 Online Training of the ESN Identifier

In this section, the online training of the ESN identifier is introduced. The ESN identifier is validation in a real-time hardware-in-the-loop simulation environment (Real Time Digital Simulator or RTDS) located at the Missouri University of Science and Technology, in Rolla, MO.

5.3.1 Online Training Approaches for the ESN Identifier

The ESN identifier is trained to predict the errors between the load current harmonics and the harmonics of the current injected by the APF. The training process consists of

two stages: in the first stage, which is called *forced training*, the ESN identifier is trained to track the Plant dynamics when the inputs to the Plant are perturbed using Pseudo Random Binary Signals (PRBS); in the second stage, called *natural training*, the ESN identifier is trained to learn the dynamics of the Plant when the PRBS is removed and the Plant is exposed to natural large disturbances such as a sudden load change. In each case the estimated output of the identifier is compared with the actual output of the Plant and the resultant error vector is back-propagated through the ESN identifier to adjust its weights. The schematic diagram of the forced training process is shown in Figure 5.4. First, the switches S_1 through S_4 are at position 1. Conventional PI controllers are used to obtain the steady-state inputs of the PLANT, namely, $v = [v_{d5}^*, v_{q5}^*, v_{d7}^*, v_{q7}^*]$. The steady state values are listed in Table 5.1 for a particular condition of the load.

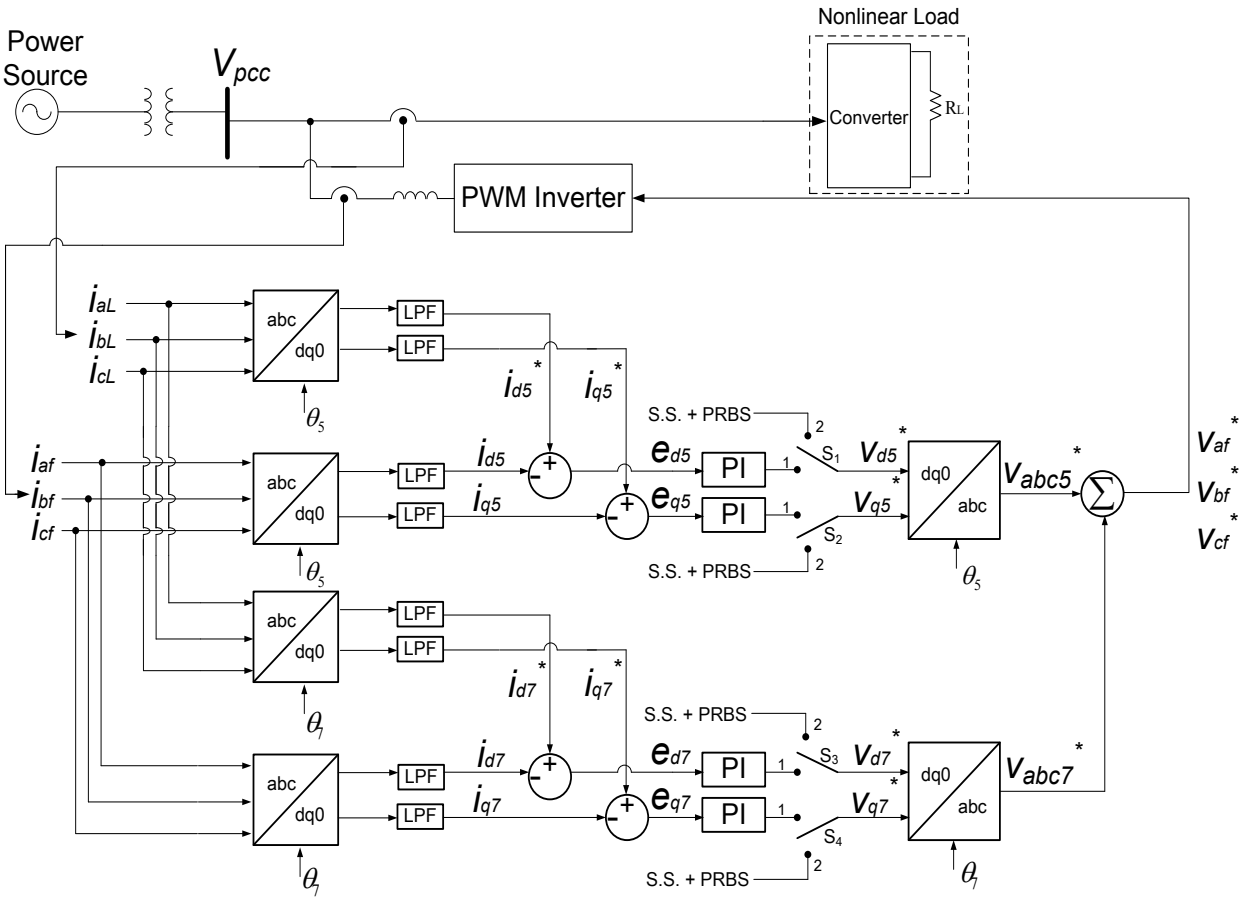


Figure 5.4: Forced training of the ESN Identifier.

Table 5.1: Steady state value of PLANT inputs.

v_{d5}*	v_{q5}*	v_{d7}*	v_{q7}*
-12.28kV	-1.64kV	9.47kV	-2.29 kV

The PLANT is then stopped and switches S1 to S4 are switched from position 1 to position 2. Under this condition, the previous steady inputs to the Plant, [v_{d5}*, v_{q5}*, v_{d7}*, v_{q7}*], are disturbed by adding pseudo-random-binary-signals (PRBSs) to the steady-state inputs. The magnitude of each injected PRBS is limited to ± 10% of its steady-state

value, and contains frequencies of 30, 60 and 90 Hz. The PRBS disturbs the system and causes small deviations of e_{d5}, e_{q5}, e_{d7} and e_{q7}, so that the ESN identifier can learn the system dynamics close to the normal operating range. Typical waveforms of disturbed v_{d5}*, v_{q5}*, v_{d7}* and v_{q7}* are shown in Figure 5.5(a) and the waveforms of the corresponding e_{d5}, e_{q5}, e_{d7} and e_{q7} caused by the disturbance are shown in Figure 5.5(b). So the vector v= [v_{d5}*, v_{q5}*, v_{d7}*, v_{q7}*] which is the input to the Plant, and the vector e_i= [e_{d5}, e_{q5}, e_{d7}, e_{q7}] which is the output from the PLANT, are used for training the ESN identifier.

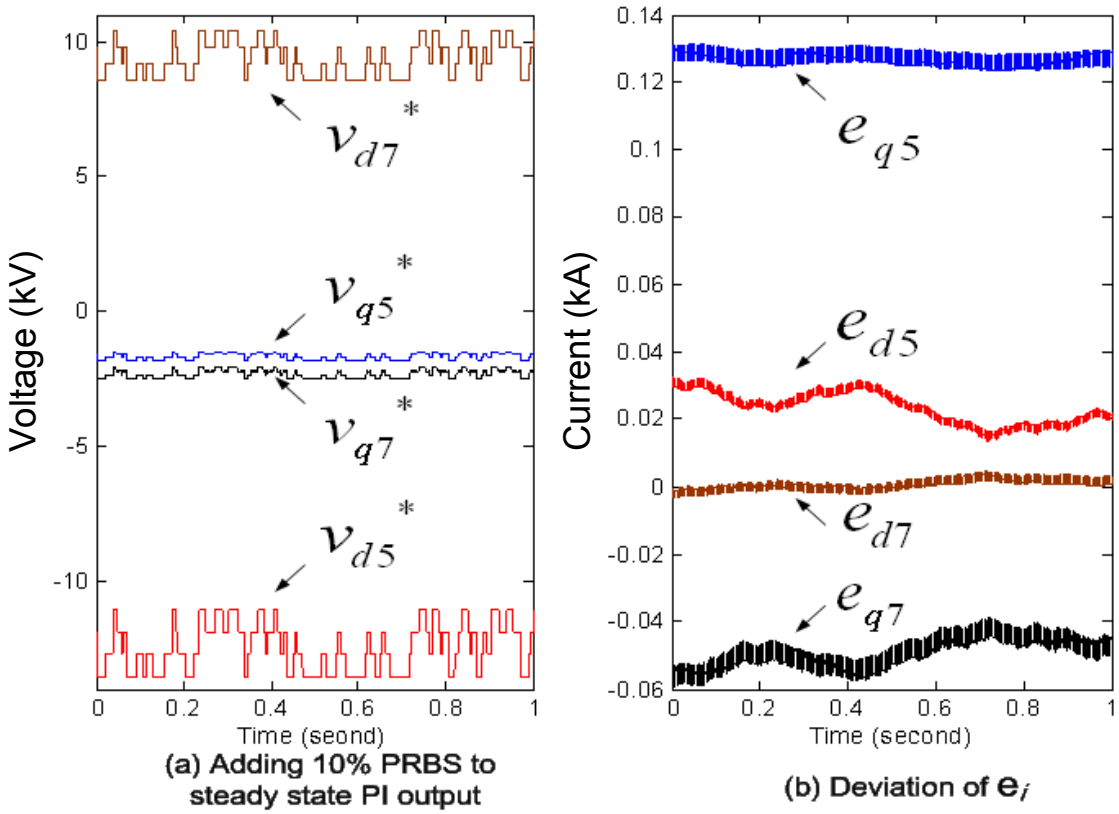


Figure 5.5: Inputs and outputs of the PLANT with PRBS disturbance.

5.3.2 Real-time Implementation of Online Training for the ESN Identifier

The ESN identifier in the proposed neurocontrol scheme is implemented on the Innovative Integration M67 card consisting of a TMS320C6701 DSP. The APF and the power system are simulated on a real-time digital simulator (RTDS) [116]. The connections and the flow of the data between the DSP and the RTDS are shown in Figure 5.6(a). First, the system is modeled with the simulation software RSCAD in a remote workstation, and then RTDS downloads from RSCAD and runs the runtime file generated by the software, finally the real time simulation values calculated by RTDS are sent to the M67 DSP card host personal computer via the DSP-RTDS interface and used for online training of the ESN identifier. A block diagram of the online training algorithm of the ESN identifier is shown in Figure 5.6(b). At each time step, the ESN identifier predicts the outputs, which is the error vector between the harmonics of the load currents and those of the current injected by the APF, with the voltages transformed by the 5^{th} and 7^{th} reference frames as the inputs. Then, the errors between the actual values and these predicted values are calculated and used to update the weights of the ESN identifier. As the entire implementation is under real-time operation, the DSP and the RTDS can emulate the actual performance of the ESN identifier in practical applications. Due to the limited number of input/output channels of the DSP card, only the 5^{th} and 7^{th} harmonics are tested under the real-time implementation environment.

The relevant inputs and outputs of the simulated power system on the RTDS are scaled by a scaling factor, because the range of the RTDS A/D channels have to be kept within [-10, 10] volts. In order to find the proper scaling factors for each channel, the RTDS simulation is run for a sufficiently long time, and the maximum absolute values of

the data are used as the scaling factors for each harmonic component. The data are scaled by these scaling factors before being sent to the output channels of the RTDS, which are the inputs to the ESN identifier. Table 5.2 lists the scaling factors for each harmonic component.

The training algorithm that is used here is the ESN online training algorithm described in Section 4.2.2.

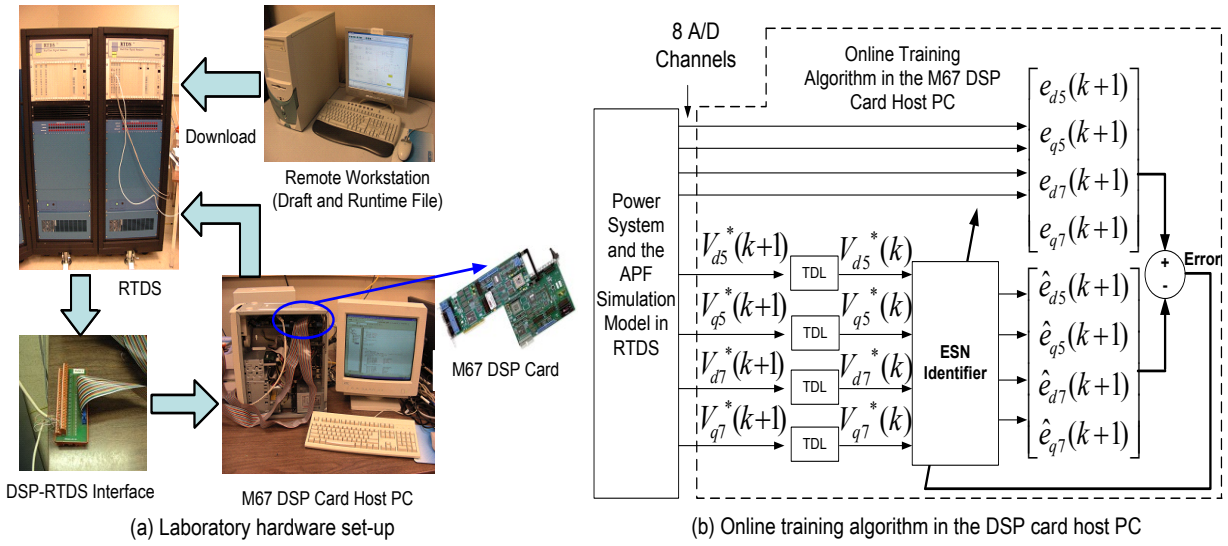


Figure 5.6: Hardware setup for the online training of the ESN identifier. TDL denotes time delay.

Table 5.2: Scaling factors of the RTDS outputs.

	5-d	5-q	7-d	7-q
Input v^*	15	1.9	12	3
Output e_i	0.2	0.333	0.05	0.2

5.3.3 Online Training Results of the ESN Identifier

The online testing results of an ESN identifier with 50 internal neurons are shown in Figure 5.7. The figure shows the four outputs estimated by the ESN identifier versus the desired values obtained from the RTDS simulation. These results show that the ESN with 50 internal neurons is capable of providing accurate modeling of the dynamic PLANT. The curve of the ESN estimated output and the curve of the simulation value agree closely in all four figures. The small error is partly caused by the limited D/A resolution of the DSP card.

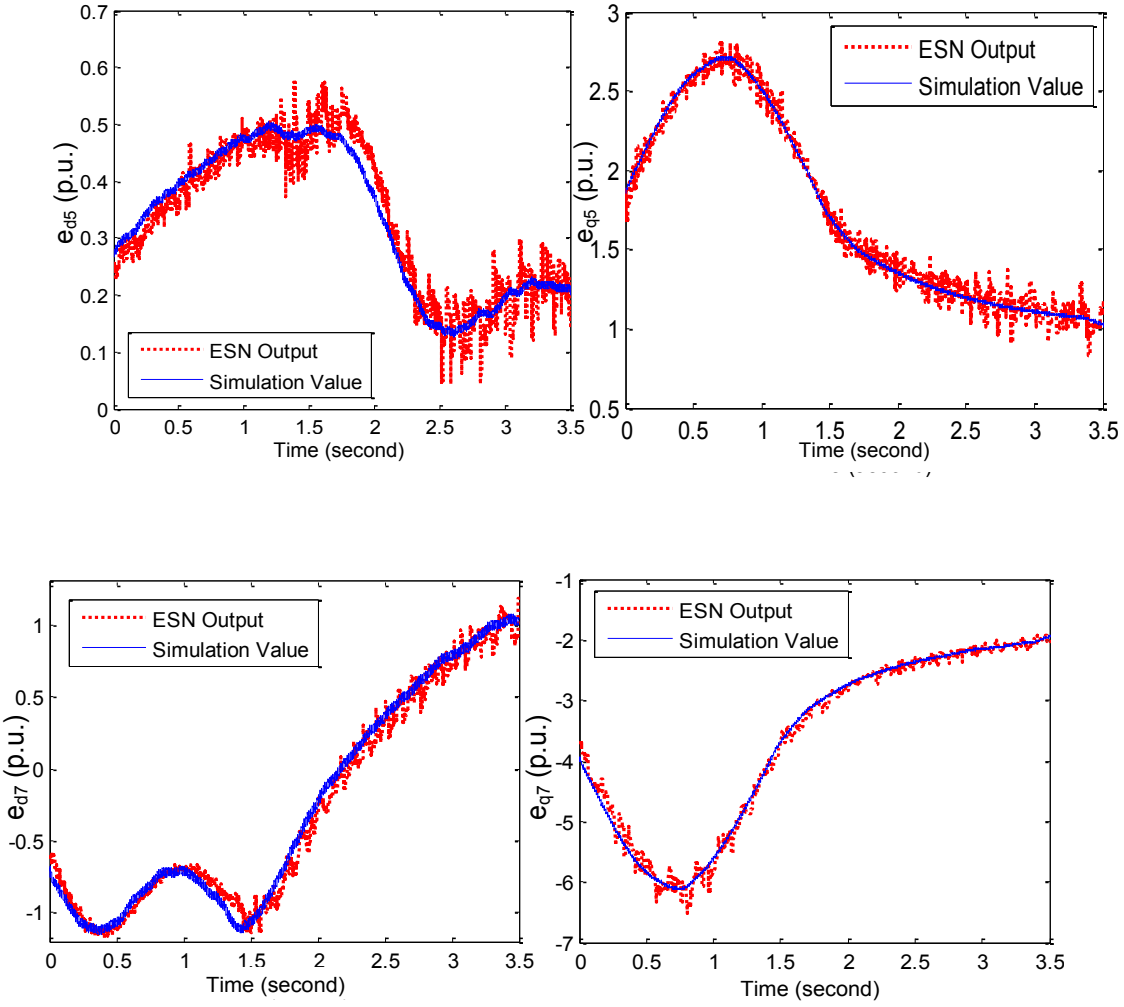


Figure 5.7: Online training results of the ESN identifier with 50 internal neurons.

5.3.4 Effect of ESN Dynamic Reservoir Sizes

To evaluate the dynamic system modeling (identification) capability of the ESN with different sizes of "dynamic reservoir", two other ESNs with 20 and 100 internal neurons respectively are tested and compared with the performance of the ESN with 50 internal neurons. For convenience, only one of the estimated outputs of the ESN identifier ($\hat{e}_{d7}$) and its desired output (e_{d7}, simulation output obtained from the RTDS) are shown in Figures 5.8(a) and (b) respectively. Figures 5.8 and 5.9 show that all three ESN identifiers are capable of providing accurate modeling of the dynamic PLANT. Comparisons of the other three outputs yield similar results.

However, although the ESN identifiers with more internal neurons yield better dynamic modeling capability, they also require more computational effort and start introducing a noticeable time lag, as shown in Figure 5.8 (b), therefore making them less acceptable as a dynamic system identifier, as fast response is highly desirable for the dynamic control of an APF.

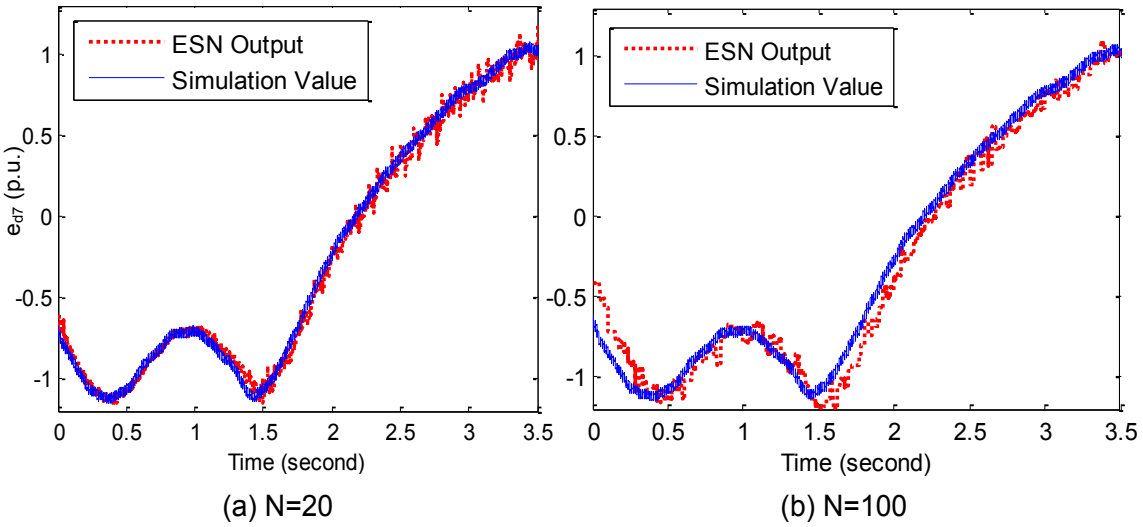


Figure 5.8: Comparison of the ESN estimated output $\hat{e}_{d7}$ and RTDS simulation value e_{d7} using different numbers of internal neurons.

The computational time required for each training step is shown in Figure 5.9(a), for the three ESNs with different internal neuron sizes, as the internal neuron size increases, the required computational time greatly increases.

The Mean Squared Errors (MSEs) of the three ESN identifiers with different internal neuron sizes are shown in Figure 5.9(b); the MSEs for the 20 and 50 internal neurons decrease much faster than for the ESN with 100 internal neurons because a much longer training process is needed for the ESN with the larger dimension of 100. Considering the shorter computational time required, the ESN identifier with 20 internal neurons is preferred over the one with 50 internal neurons. The accuracies of the ESN identifiers as well as their learning capability and speeds are highly dependent on ESN parameters and initial conditions. However, all MSEs converge and stabilize to a low number, which indicates all three ESN identifiers are able to identify the system dynamics.

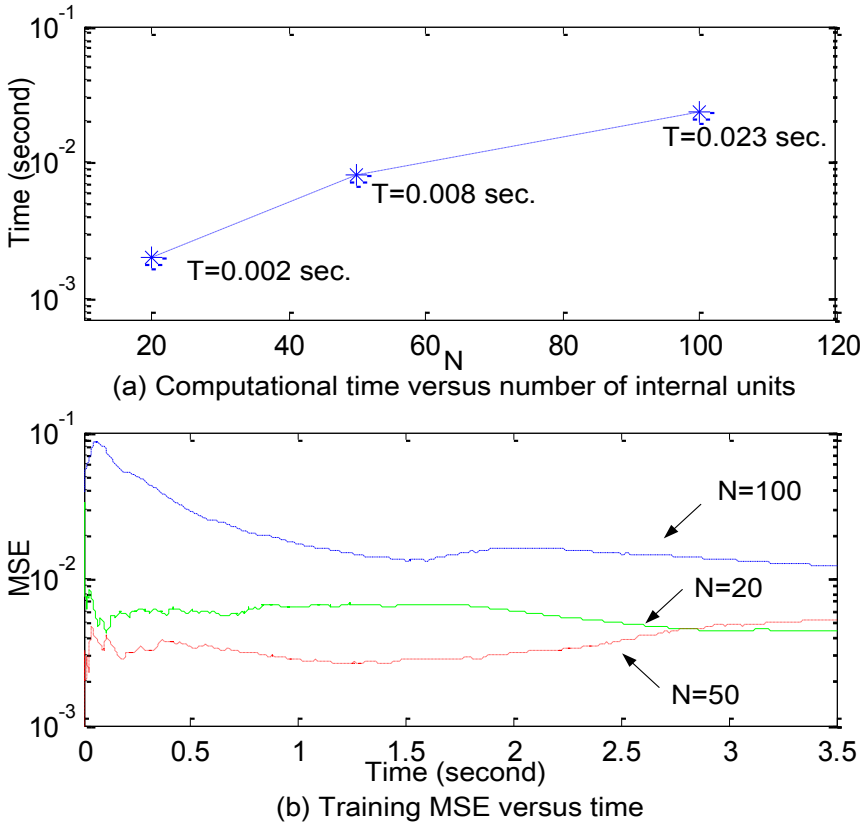


Figure 5.9: Comparisons of computational efficiencies and training MSEs.

For the real-time implementation of the ESN identifiers, deciding upon the number of internal neurons is a tradeoff between identification accuracy and required computational time, given the computational capability of the implementation hardware and system complexity.

5.4 Online Training of the ESN Controller

After the training of the ESN identifier, the ESN controller can then be trained and tested so that the overall indirect adaptive control scheme can be realized. The ESN control system is developed using PSCAD due to limited computational capability of the

real-time hardware-in-the-loop simulation setup on the RTDS system. Hence all the results presented in this section are obtained using PSCAD.

5.4.1 Training algorithm of the ESN Controller

The training of the ESN controller in Figure 5.4 consists of two stages: (1) offline pre-training using PI controller input and output; (2) online training. The purpose of the pre-training is to make sure that the ESN controller can at least work like the conventional PI controllers. Then the ESN identifier and controller are exposed to various load conditions during the online training process so that they can learn the system dynamics and act adaptively. The pre-training is done in MATLAB using simulation data from PSCAD. The weights obtained from the pre-training are then used as the initial weights of the ESN controller.

5.4.2 Performances Comparison of the ESN Controller and PI Controllers

The control result of the proposed indirect adaptive control scheme under five different converter firing angles is shown in Figure 5.10 when a change in firing angle occurs every 0.5 seconds. Only the waveforms of e_{d5} are plotted here, and e_{q5}, e_{d7}, e_{q7} behave in a similar manner. A comparison of the ESN controller performance and the PI controller performance is also given in Figure 5.10. ESN controllers show faster damping and smaller overshoot than PI controllers. It takes less time for the ESN controller to drive e_{d5}, e_{q5}, e_{d7}, e_{q7} to zero, which is highly desirable in this active filter application.

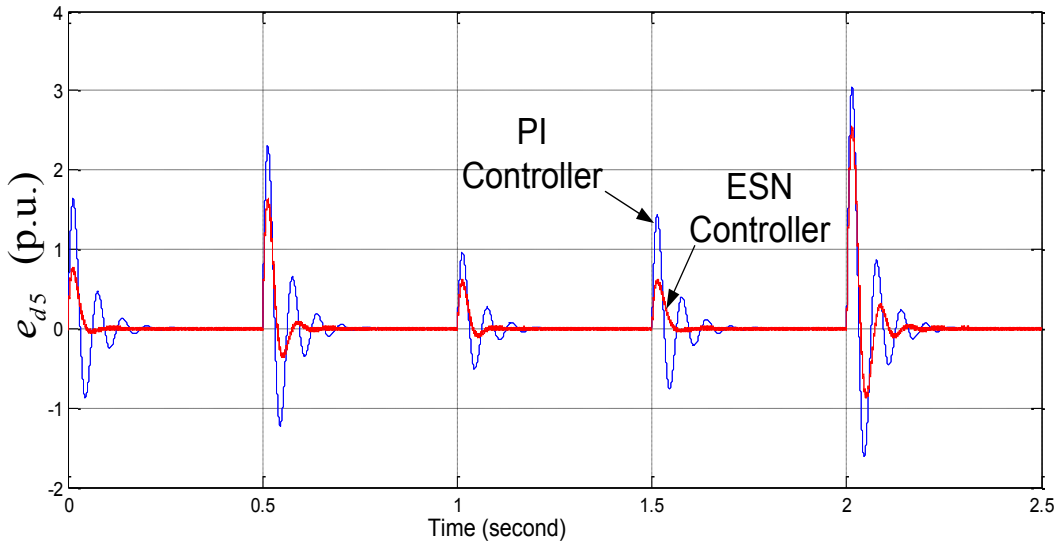
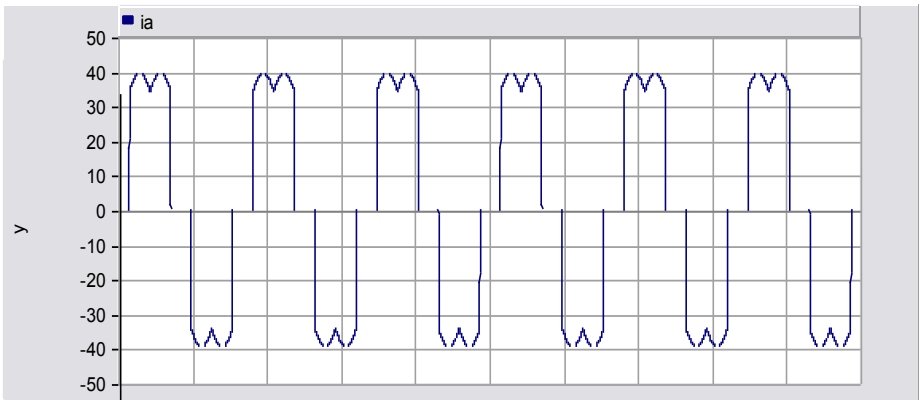


Figure 5.10: Comparison of ESN controller and PI controller performances.

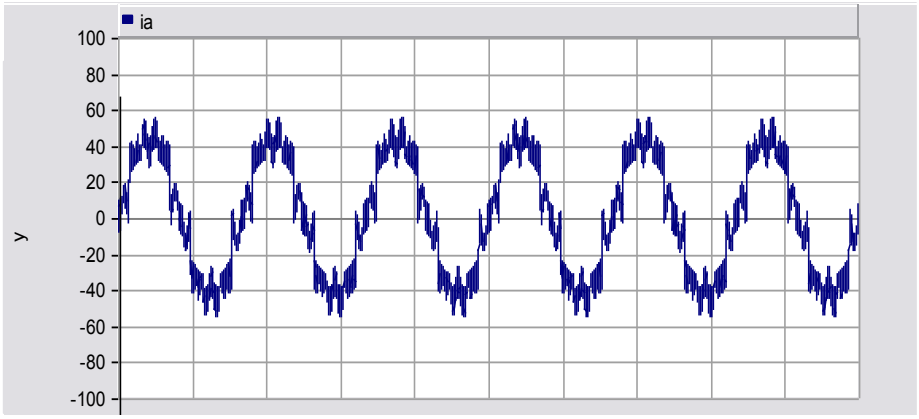
5.4.3 Source Current Waveforms

The phase A source currents before (Figure 5.11(a)) and after (Figure 5.11(b)) the harmonic compensation are shown below. The harmonic current injected by the active filter in phase A is also shown in Figure 5.11(c). This figure only shows the performance of the ESN controller under one load condition (Firing Angle=0°. 0.1 second data is plotted.)

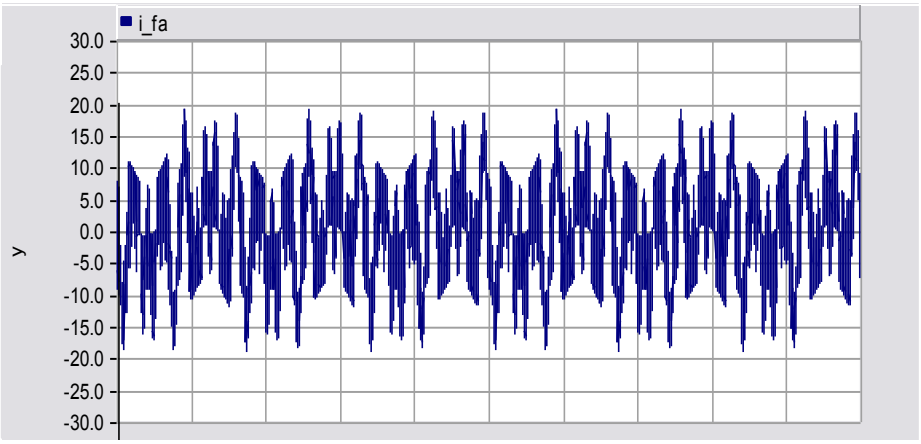
From Figure 5.11(a) and (b) it can be seen that the current waveform coming out of the power source has less harmonics after the harmonic compensation. The total harmonic distortion (THD) is reduced from 31.08% to 7.82%. The remaining harmonics in Figure 5.11 (b) could be removed by adding 11^{th} and 13^{th} (and so on) order filter stages to the 5^{th} and 7^{th}.



Phase A source current i_a without the active filter (THD=31.08%).



(b) Phase A source current i_a with the active filter (THD=7.82%).



(c) Phase A current i_{fa} injected by the active filter.

Figure 5.11: Source current and active filter injected current.

5.5 Chapter Summary

An indirect adaptive neurocontrol scheme which uses two ESNs to control a shunt active filter has been proposed in this chapter. As the first step in the proposed neurocontrol scheme, the feasibility of the ESN identifier has been evaluated by simulating the system in real-time hardware-in-the-loop simulation. The testing results of the ESN identifier have shown that the ESN is capable of providing fast and accurate system identification for the indirect neurocontrol of a shunt active filter, even when the load condition changes nonlinearly. Furthermore, the ESN controller has been pre-trained first and then online trained. The performance of the proposed indirect adaptive control scheme has been compared with traditional PI controllers. Since PI controllers are only tuned around a certain operation condition, when the load changes nonlinearly, the performances of the PI controllers degrade. The ESN controller used in the indirect adaptive control has been trained under different load conditions and actually learns the complicated system dynamics of the plant, so it gives better control results than the PI controllers.

It has been shown in Chapter 4 and Chapter 5 that the reservoir-computing-based ANNs, i.e., ESNs, have strong modeling capability with limited computational effort needed, which can be used for various modeling and control applications. The reservoir-computing approach in the ESN can be introduced to assist the development of biologically-inspired artificial neural networks, which more closely emulate the neurons in the brain than conventional artificial neural networks. This is introduced in the next chapter.

CHAPTER 6 BIOLOGICALLY INSPIRED ARTIFICIAL NEURAL NETWORKS

6.1 Overview

Despite the wide range of applications of ANNs in monitoring, operation and control of power systems and their effectiveness compared to conventional techniques, it is believed that in vivo LNNs have much larger computational potential than ANNs. However, as discussed in Chapter 1 and Chapter 3, a key question remains unanswered: how to utilize the computational potential in LNNs.

To bridge the gap between ANNs and LNNs, a new type of ANN, i.e. the Biologically-Inspired Artificial Neural Network (BIANN) is proposed in this chapter. A BIANN processes information in a more "brain-like" fashion than the traditional ANN. The mean firing rate (MFR) coding and decoding method and the Izhikevich model [117] for spiking neurons are used in the BIANN architecture. An online training algorithm of the BIANN based on reservoir computing is proposed and validated. The proposed BIANN is illustrated in Chapter 7 in the online identification of a single machine infinite bus (SMIB) power system, to predict the multi-step-ahead response of that system and to validate the feasibility of the online training algorithm of the BIANN.

This Chapter describes the BIANN in detail. The proposed BIANN is proposed based on spiking models of living neurons, with consideration of neuronal latency plasticity for network connections. A simplified encoding and decoding method is proposed to encode inputs to BIANNs and read information from BIANNs, and their modeling capability of proposed BIANN is validated through simulation. These proposed BIANNs have the capability of modeling dynamic systems. The BIANNs can

also potentially be used for the training of LNNs for future development of computational intelligence.

6.2 Modeling of LNNs using the Izhikevich Model

Among all spiking models of living neurons, the Izhikevich model [117], as described in Chapter 3, has the capability of representing various behaviors of a living neuron, without involving complex computational effort. Therefore, this neuron model is used as the model of LNNs in the BIANNs.

6.2.1 Composition of the Spiking Neuron Network

The network consists of two types of neurons: namely excitatory neurons and inhibitory neurons. Each type has its own neuron model to represent different neuron behaviors. The ratio of excitatory to inhibitory cells is typically 4 to 1, as in the mammalian neocortex [117]. The probability of connection is 10% for all the neurons in the network. This means that each excitatory neuron is assumed to be randomly connected to 10% of the neurons in the entire neural network, and each inhibitory neuron is connected to excitatory neurons only, as in the neocortex. Each connection is featured by a synaptic weight and a conduction delay, which are discussed in the following sections.

6.2.2 Izhikevich Spiking Neuron Model

Each neuron in the network is described by a simple spiking model as follows:

$$\begin{cases} \frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I \\ \frac{du}{dt} = a(bV - u) \end{cases}$$

$$\text{if } V \geq 30 \text{ mV, then } \begin{cases} V = c \\ u = u + d \end{cases}, \tag{6.1}$$

where *a*, *b*, *c* and *d* are parameters that represent various behaviors of living neurons.

The variable *V* represents the membrane potential of the neuron, and *u* represents a membrane recovery variable, which accounts for the activation of K+ ionic currents and inactivation of Na+ ionic currents, and it provides negative feedback to *V*. After the spike reaches its apex at +30 mV, which is not to be confused with the firing threshold, the membrane voltage *V* and the recovery variable *u* are reset according to (6.1).

The *a,b,c,d* parameters are set differently for excitatory neurons and inhibitory neurons. Corresponding to cortical neurons, the parameters for excitatory neurons are set as [*a, b, c, d*] = [0.02, 0.2, -65, 8]; while for inhibitory neurons, [*a, b, c, d*] = [0.1, 0.2, -65, 2]. For the design of the BIANN, all these parameters are randomly altered in a ±10% range to generate more patterns of neuron behaviors [102].

Assuming a time step of 1 ms, (6.1) can be rewritten as the following iterative form:

$$\begin{cases} V(n+1) = 0.04V^2(n) + 6V(n) + 140 - u(n) + I(n) \\ u(n+1) = u(n) + a[bV(n) - u(n)] \end{cases} \tag{6.2}$$

6.2.3 Spike Timing-Dependent Plasticity

Each connection between two neurons (e.g. connection from neuron *i* to neuron *j*) has two properties: namely the synaptic weight (S_{ij}) and the conduction delay (D_{ij}). The synaptic connections in the model are modified according to the following spike-timing-dependent plasticity (STDP) rule [118]:

If a spike from an excitatory presynaptic neuron arrives at a postsynaptic neuron (possibly making the postsynaptic neuron fire), then the synapse is potentiated

(strengthened). In contrast, if the spike arrives right after the postsynaptic neuron has fired, then the synapse is depressed (weakened). [114]

This STDP rule is illustrated in Figure 6.1.

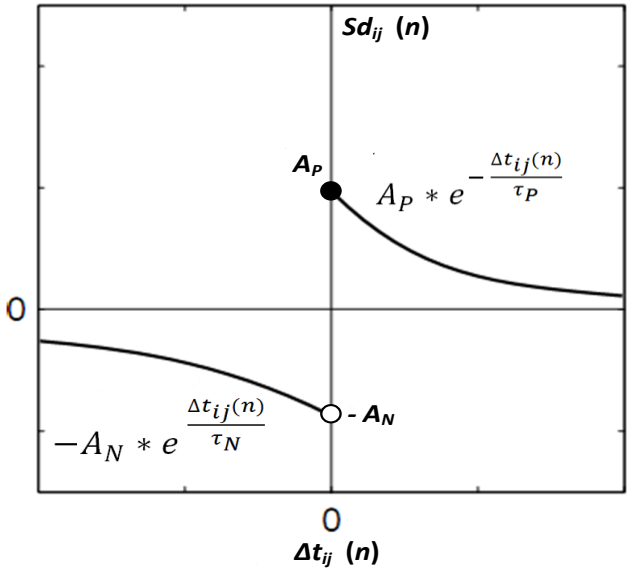


Figure 6.1: STDP rule.

Define $\Delta \mathbf{t}_{ij}$ (**n**) as the time difference between the firing of postsynaptic neuron (neuron *j*) and the arrival of excitatory spikes from a presynaptic neuron (neuron *i*). Let Sd_{ij} be the change of synaptic weight, then Sd_{ij} can be calculated using the following equation [117]:

$$\mathbf{Sd_{ij}\ (n)} = \begin{cases} A_P * e^{-\frac{\Delta t_{ij}(n)}{\tau_P}} & if\ \mathbf{\Delta t_{ij}\ (n)} \geq 0 \\ -A_N * e^{\frac{\Delta t_{ij}(n)}{\tau_N}} & if\ \mathbf{\Delta t_{ij}\ (n)} < 0 \end{cases} \tag{6.3}$$

where A_P is the maximum magnitude of synaptic weight increase; A_N is the maximum magnitude of weight decrease; τ_P and τ_N are time constants. The commonly used parameter setting is $A_P = 0.1$; $A_N = 0.12$; $\tau_P = \tau_N = 20$ ms [113].

6.2.4 Maturing of Spiking Neural Networks

Due to the neuronal latency plasticity of spiking neural networks, although initialized randomly, using STDP rules, the synaptic weights in a spiking neural network will eventually distribute in two ranges: 0-10% and 90%-100% of the weight limits given a certain input. If more than 40% of all the synaptic weights fall in these two ranges respectively, the spiking neural network has matured. Therefore, it is important to first mature the spiking neural network before it can be used in the BIANN.

If BIANN is to be used to model (or identify) a nonlinear system, for maturing the BIANN, it is preferred to have the actual inputs of this nonlinear system that need to be modeled as the inputs to the spiking neural network so that the spiking neural network can mature with the same input patterns as the system inputs. When such conditions cannot be met, random spiking signals can be used for maturing spiking neural networks.

6.3 From ESN to BIANN

The ESN has been shown in chapter 4 to have strong modeling capabilities with low computational effort required. This capability lies in the dynamic reservoir, which can provide various patterns of output following any input signal. Meanwhile, its low computational requirement is due to its training scheme: only the output weights are updated during training.

Due to the complexity of LNNs, the spiking neuron models have a strong potential to generate various output patterns given a certain input signal, similar to the dynamic reservoir in ESNs. The concept of reservoir computing is also preferred in order to reduce the required computational effort in BIANNs for modeling of dynamic systems.

The reservoir-computing approach is introduced to the BIANN in order to obtain strong modeling capability with low computational requirements. With the adoption of the Izhikevich spiking neuron model, one of the major challenges is the encoding of the input signal to the spiking neuron network.

6.3.1 Input Layer and Encoding Algorithm

The major difference between conventional ANNs and LNNs lies in the format of information transmission: in ANNs, continuous analog signals are transmitted among neurons; while in LNNs, spiking signals are transmitted among neurons. Therefore, the encoding approach, which converts the analog input signals to a spiking signal, is essential for adopting spiking neuron models into BIANNs. In this proposed BIANN, the mean firing rate (MFR) approach is adopted as the encoding and decoding approach due to its simplicity. The encoding procedure is described as follows.

Scale each original signal to be in the range of [0.1 0.9] using (6.4);

$$input_{scaled}(n) = \frac{input(n) - \min\,(input(n))}{\max(input(n)) - \min\,(input(n))} \times 0.8 + 0.1 \tag{6.4}$$

At time step n, calculate the mean firing rate of all the M spike trains over the period (n-T+1) to (n-1) using (6.5);

$$past(n) = \frac{1}{T \cdot M} \sum_{i=1}^{M} \sum_{\tau=n-T+1}^{n-1} Fire_i(\tau) \tag{6.5}$$

Here, T is the time window for calculating the mean firing rate (T=20 ms) and $Fire_i(\tau)$ is the spiking signals.

Calculate the difference between $past(n)$ and $input_{scaled}(n)$ using (6.6);

$$need(n) = T \cdot M \cdot [input_{scaled}(n) - past(n)] \tag{6.6}$$

At time step n, calculate how many spikes in total still need to be generated to form $input\_scaled\ (n)$ using (6.7);

$$\sum_{i=1}^{M} Fire_i(t) = \max(\min(round[need(n)], M), 0) \quad (6.7)$$

The result from this step is an integer in the range of [0, M].

Assign the spikes that still need to be generated at time step n randomly to the M converting units.

An illustration of the encoding method is shown in Figure 6.2. At each time step n, the value of the scaled signal $input\_scaled\ (n)$ is considered as the mean firing rate of all the M spike trains over the time period [(n-T+1), n]. Each original continuous signal is encoded into M spatiotemporal spike trains, which are then sent into M excitatory neurons in the BIANN as input, i.e. "$I(n)$" in (6.2).

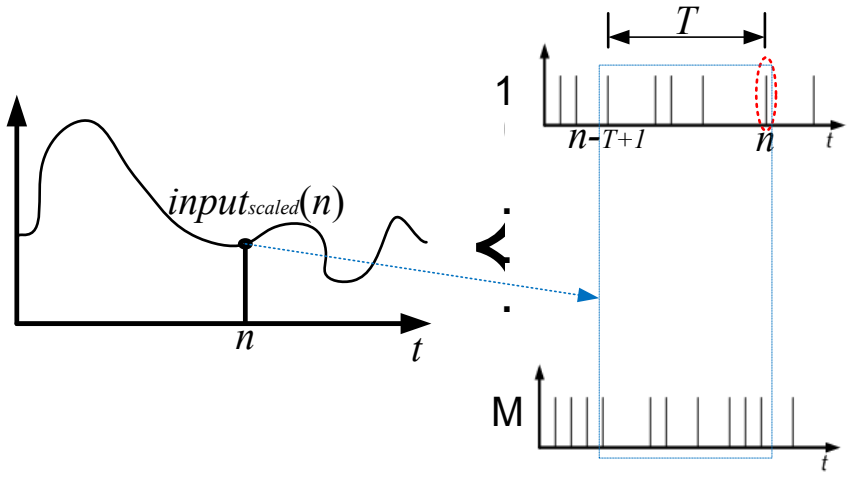


Figure 6.2: Illustration of the encoding method.

The same encoded spiking signals are fed to multiple input neurons repeatedly in the Izhikevich spiking neuron model. For instance, assuming 100 input neurons are selected in the spiking neuron model, if an analog input signal is encoded as 20 spiking signals,

each spiking signal is given to 5 excitatory neurons so that the input spiking signal can generate enough input to the spiking neuron model.

6.3.2 Decoding Algorithm

The computation in the dynamic reservoir strictly follows the Izhikevich spiking neuron model described in Section 6.2. When a neuron fires at time step n, the output of this neuron at this moment is marked as $Fire_i(n)=1$, otherwise, $Fire_i(n)=0$.

The decoding method is the well-known mean firing rate method [84]. At time step n, the output of the i^{th} neuron in the BIANN is decoded from spiking signals $Fire_i$ to continuous signals using (6.8):

$$output_i(n) = \frac{1}{T}\sum_{\tau=n-T+1}^{n} Fire_i(\tau) \tag{6.8}$$

Again $T=20$ ms is the averaging window. $output_i(n)$ is not the final output of the BIANN, since an additional readout layer with trainable weights is still needed to extract useful information from the dynamic reservoir.

6.3.3 Readout Layer and Training Algorithm

The readout layer of the BIANN is a set of trainable weights. Unlike the spontaneous, STDP-based weights' update in the dynamic reservoir, these weights are adjusted using supervised learning. The final output of the BIANN is calculated using (6.9):

$$\hat{y}(n) = f[W^{out} \cdot output(n)] \tag{6.9}$$

where $\hat{y}(n)$ is the output of the BIANN at time step n, W^{out} is the output weight matrix and $output(n)$ is the decoded analog signal from all the neurons in the spiking neuron network, including the input neurons. The function f can be a linear function, a sigmoid function. In this thesis, a linear function is used.

Similar to ESNs, in BIANNs, only the output weights are updated at each time step to reduce the required computational effort. Various training algorithms can be applied to BIANNs, such as the pseudo-inverse approach and backpropagation approach.

For pseudo-inverse training, some duration of the desired BIANN output and the decoded output of the spiking neuron network are recorded for computing the optimal output weights for modeling dynamic systems:

$$W^{out}(n) = Teacher(n) \cdot pseudoinverse(output(n)). \quad (6.10)$$

output(n) is the decoded output signal of the dynamic reservoir or spiking neuron model; *Teacher(n)* is the desired output of the BIANN. For a given system, the pseudo-inverse approach can provide the optimal modeling of the system assuming no change in the behavior of the system. However, for dynamic systems, the backpropagation algorithm is preferred for its quick adaptation and learning of any system change which may happen over time.

At each time step, the readout weights are updated as:

$$error(n-1) = y(n-1) - \hat{y}(n-1) \quad (6.11)$$

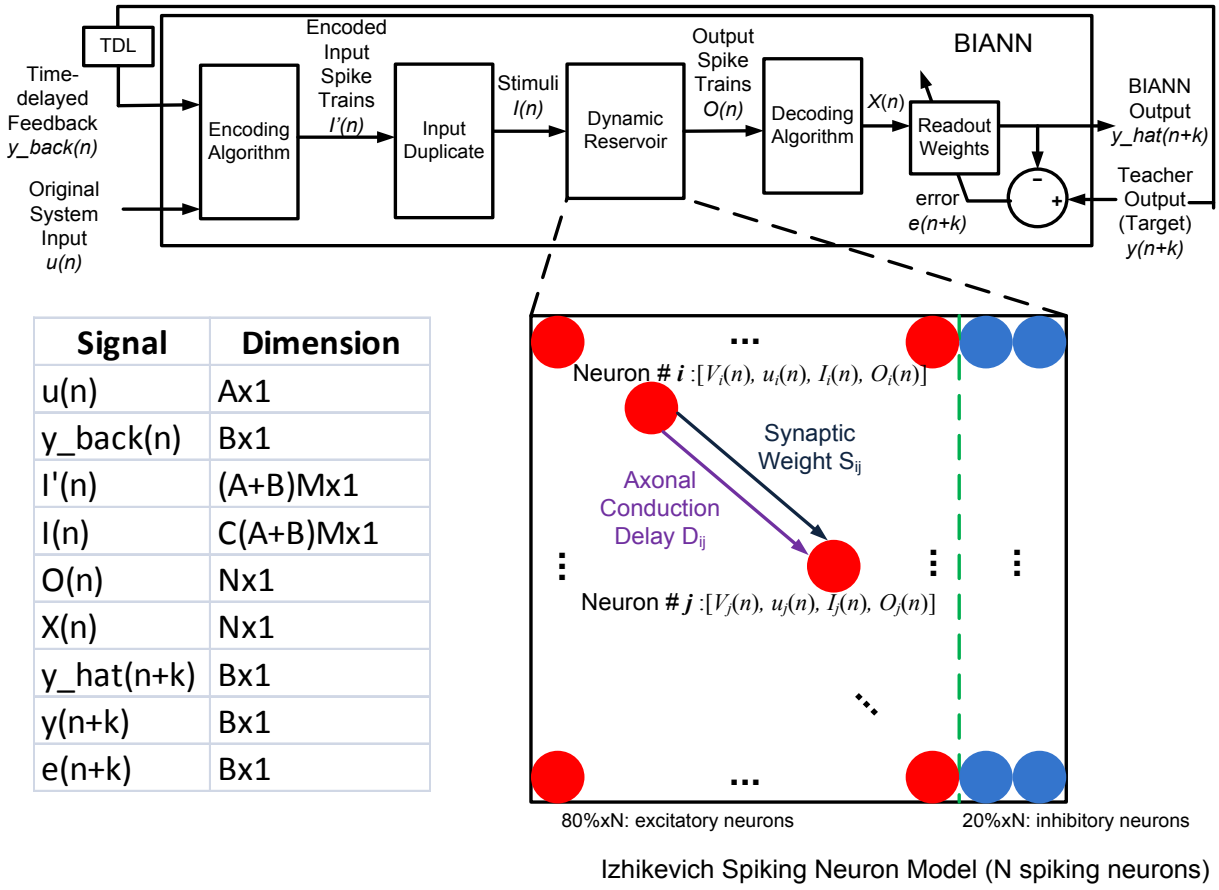
$$W^{out}(n) = W^{out}(n-1) + \eta \cdot output(n-1)^T error(n-1) + \gamma \cdot output(t-2)^T error(n-2) \quad (6.12)$$

where η and γ are the learning gain and the momentum gain, respectively.

For modeling static systems, the pseudo-inverse approach is preferred to more accurately model the system behavior; while for modeling dynamic systems, the backpropagation approach is preferred to adapt to variations of the dynamic system over time.

6.3.4 *Overall Structure of BIANN*

The overall structure of a BIANN is shown in Figure 6.3.



Signal	Dimension
u(n)	Ax1
y_back(n)	Bx1
I'(n)	(A+B)Mx1
I(n)	C(A+B)Mx1
O(n)	Nx1
X(n)	Nx1
y_hat(n+k)	Bx1
y(n+k)	Bx1
e(n+k)	Bx1

Figure 6.3: Overall scheme of a BIANN.

The original continuous input signal, for example, the terminal voltage of a generator in a power system, is first encoded into spike signals. To form a bigger stimulation to the dynamic reservoir, the encoded input spike signal is duplicated depending on the number of the input signals and the number of neurons in the dynamic reservoir. These encoded stimuli are then fed to a certain number of excitatory neurons in the spiking neural network, which are randomly chosen as the input neurons. The dynamic reservoir

consists of spiking neurons. After the calculation in the dynamic reservoir, the outputs of all the neurons in the dynamic reservoir are converted back into analog signals at each time step using a decoding method. Finally, the output of the BIANN is calculated based on the readout weights. The variables in the BIANN are demonstrated in Figure 6.3 with their dimensions listed.

6.3.5 Using a BIANN for Online Modeling of Dynamic Systems

For the online modeling of a dynamic system using a BIANN, it is preferred to have a feedback input to the BIANN, shown as the optional connection in Figure 6.4. This system feedback can largely improve the ability of the BIANN for modeling dynamic systems.

The typical scheme of using the BIANN for modeling dynamic system or plant is given in Figure 6.4. The actual input of the system is first encoded into spiking signals. The spiking neural network in BIANN is then matured following the criteria given in Section 6.2.4. When the spiking neural network is matured, the outputs of all the neurons in the spiking neural network can be decoded into analog signals. At each time step, the output weights are updated using (6.12). The output of the BIANN is then computed using equation (6.9).

Since only the output weights are updated at each time step, this training approach of the BIANN is computationally efficient. Meanwhile, since reservoir-computing approach is used, the BIANN can utilize the modeling/computational capability of the spiking neural networks for modeling complex systems.

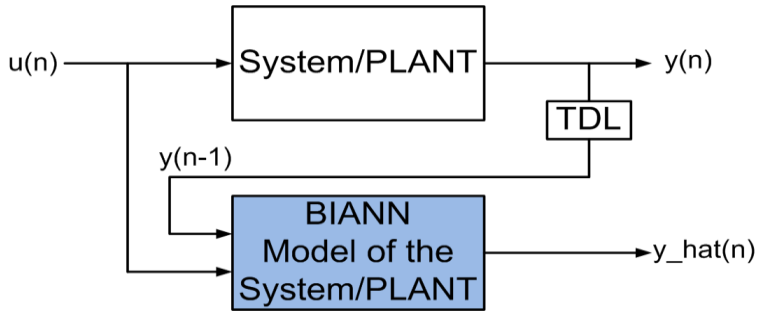


Figure 6.4: Online modeling of dynamic system using BIANN.

6.4 Simulation Validation of BIANN for Modeling Purposes

6.4.1 Simulation Setup

To show the effectiveness of the proposed BIANN, a BIANN consisting of a spiking neural network with 1000 neurons is simulated to model a single-input single-output system. 800 neurons in the spiking neural network are excitatory neurons while the remaining 200 neurons are inhibitory neurons. 100 of the excitatory neurons are randomly chosen as the input neurons. The parameters in the spiking neural network are set as suggested in Section 6.3. The learning gain and the momentum gain values are set as 0.02 and 0.01 respectively. Each input signal is encoded into 10 spiking signals. The time periods for encoding and decoding are both 20 ms. The input signal is encoded into 10 spiking signals and each input spiking signal is fed to 10 input neurons. For this example, the input and desired output are defined by (6.13) and (6.14), as shown in Figure 6.5.

$$u\,(t) \;=\; 0.5 + 0.5 * sin\,(t/100) \tag{6.13}$$

$$y\,(t) \;=\; u\,(t) \;+ 0.25 * sin\,(t/50) \tag{6.14}$$

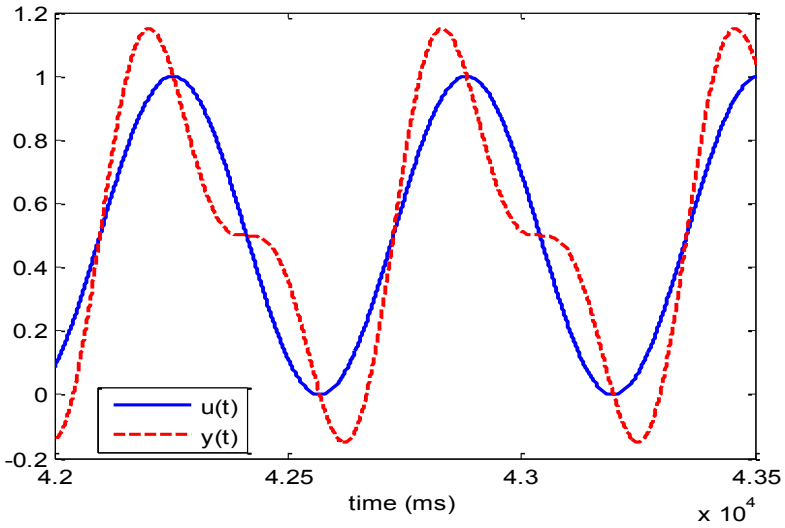
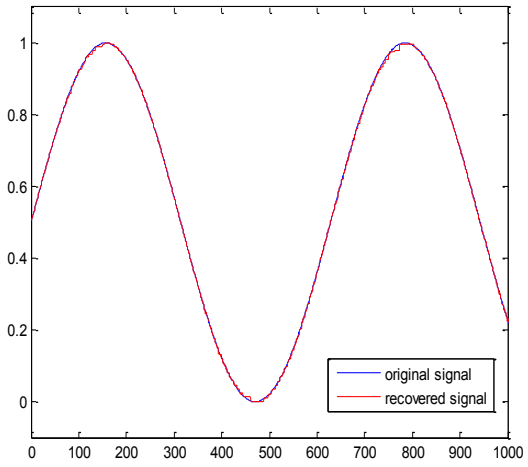


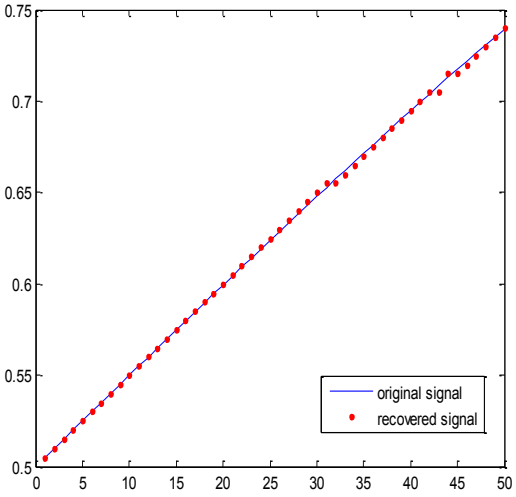
Figure 6.5: Input and desired output for modeling using BIANN.

6.4.2 Encoding of the Input Signal

To illustrate the encoding and decoding approach in the BIANN, the input signal is first encoded into 10 spiking signals. The spiking signals are then recovered using the decoding approach described in Section 6.3.2. The input signals $u(t)$ and the recovered signal $u'(t)$ are illustrated in Figure 6.6(a). It can be observed that using the MFR approach, the analog input signal is encoded with high resolution and little information loss.



(a)Original signal and recovered signal



(b)Partial enlarged view of (a)

Figure 6.6: Input signal and the recovered signal from spiking signals.

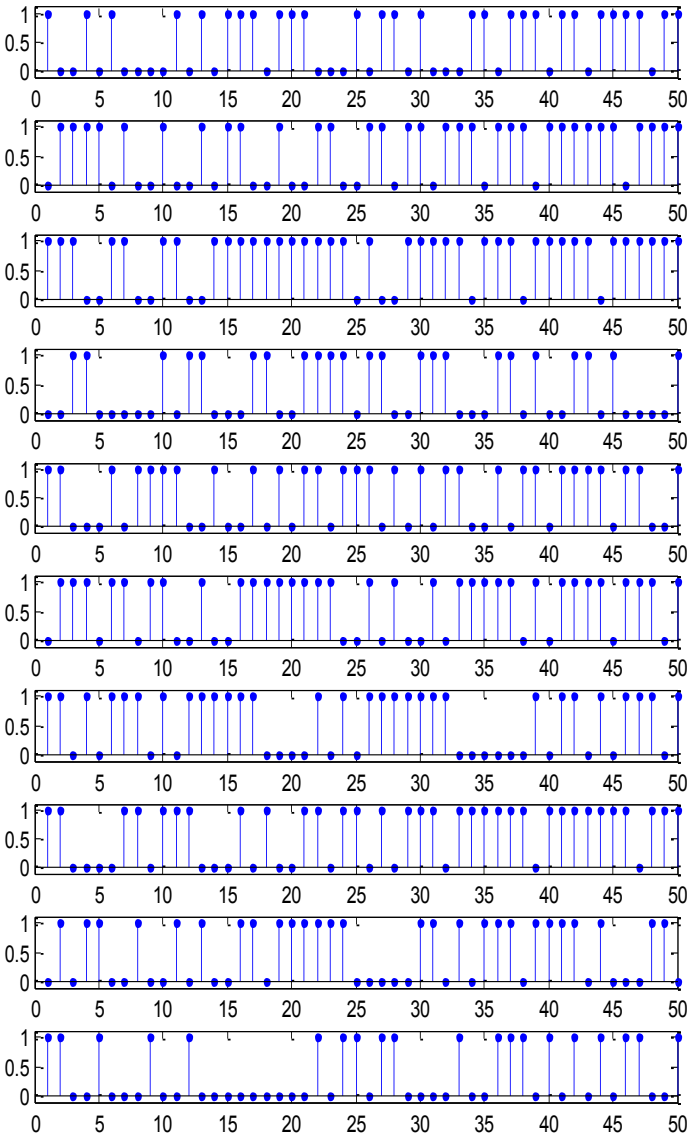


Figure 6.7: Encoded spiking signals.

The first 50 ms of data is also shown in Figure 6.6 (b) which shows that the error between the original signal and the recovered signal from spiking signals is very small. The encoded spiking signals are illustrated in Figure 6.7. The encoding algorithm has

already been described in detail in Section 6.3.1. The input signal $u(t)$ is transformed in to M spike trains, here, M=10.

6.4.3 Maturing of the Spiking Neural Network

The synaptic weights among neurons in the spiking neural network are first randomly generated as defined in Section 6.2. The encoded input signals are then continuously fed to the spiking neural network so that the synaptic weights can be matured. The process of maturing is illustrated in Figure 6.8. After roughly 200 seconds, it is observed that more than 40% of the synaptic weights lie in the 0-10% range and the 90%-100% range. After maturing, the spiking neural network can be used for modeling of dynamic systems.

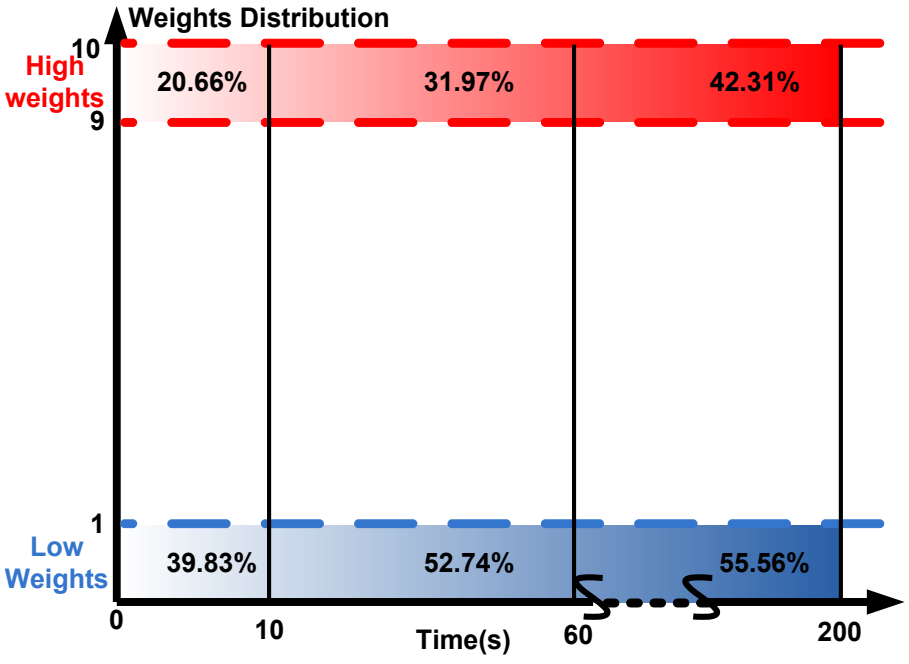
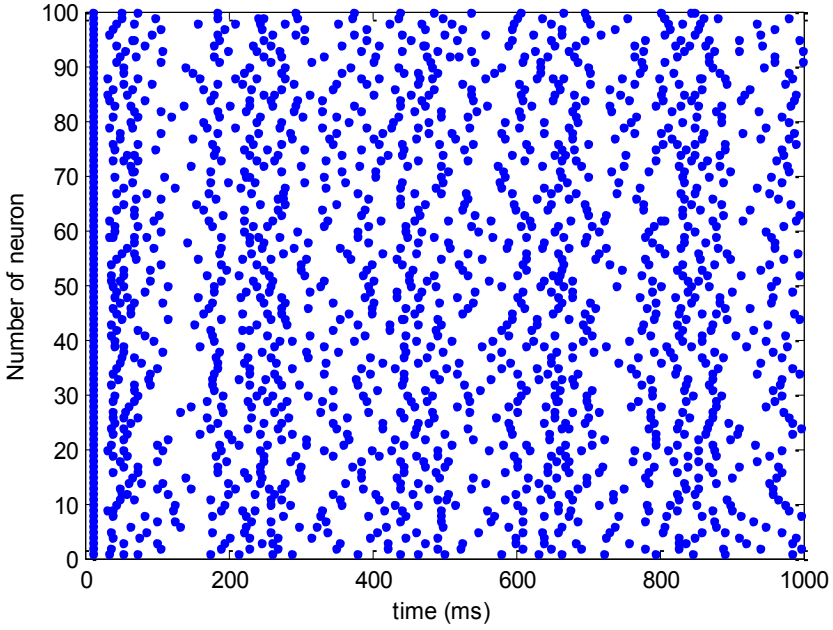


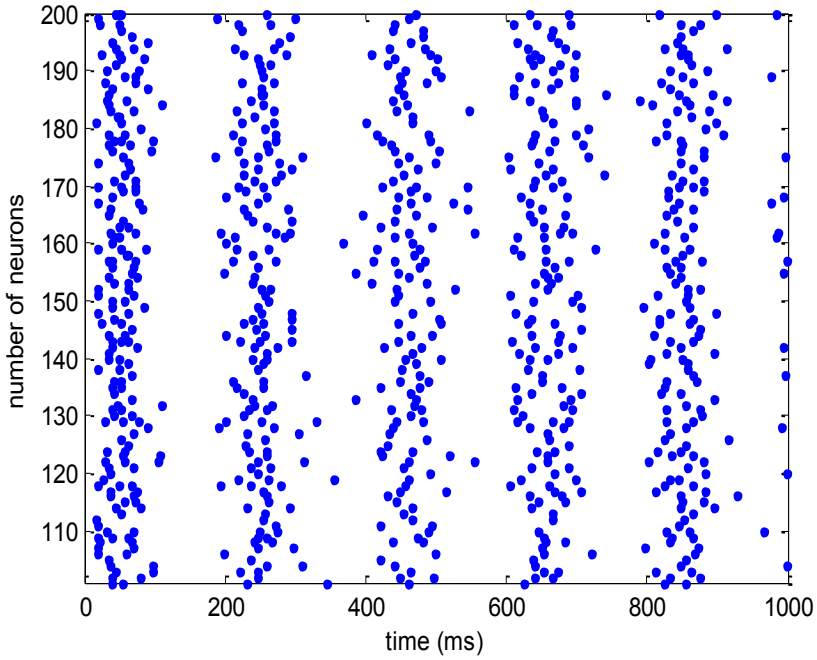
Figure 6.8: Maturing of spiking neural network.

6.4.4 Firing Patterns of the Spiking Neural Network

After maturing, the spiking neuron network can be used for online modeling of the system. When the encoded input signals are given to the input neurons in the spiking neuron network, different firing patterns are generated in the spiking neuron network, similar to the dynamic reservoir of the ESN. Such firing patterns are illustrated in Figure 6.9. Figure 6.9(a) shows the firing patterns of the 100 input neurons and Figure 6.9(b) shows the firing patterns of another 100 internal excitatory neurons. Each dot indicates a firing of a specific neuron at the corresponding time step. Although the whole spiking neural network is randomly generated, the firing pattern of the 100 input neurons are similar to each other, while the firing pattern of the other 100 internal neurons is different from that of the input neurons. This indicates that different firing patterns are automatically generated in the spiking neural network in a similar way as the dynamic reservoir in ESNs. Therefore, such various firing patterns can represent different characteristics of a system, which enables modeling of dynamic systems using BIANNs.



(a) firing pattern of 100 input neurons (neuron #1 - 100).



(b) firing pattern of 100 internal neurons (neuron #101 - 200).

Figure 6.9: Firing patterns of spiking neural network.

6.4.5 Online Modeling Results using BIANN

With the spiking neuron network excited by the encoded input signal, the output of each neuron in the spiking neural network is then decoded. The backpropagation-based training approach is then applied using (6.13). Again, during the entire modeling process, only the output weights are updated, which guarantees the computational effectiveness of BIANNs. The estimated system output and the desired system output are illustrated in Figures 6.10 and 6.11. Figure 6.10 shows the online modeling results of a BIANN without output feedback; while Figure 6.11 shows the results with output feedback. The BIANN with output feedback outperforms the BIANN without output feedback. The MSE is reduced from 0.0091 to 0.0024 calculated between 0 and 50 seconds by adding the output feedback. This is due to the fact that output feedback can provide more information of the dynamic system for online modeling.

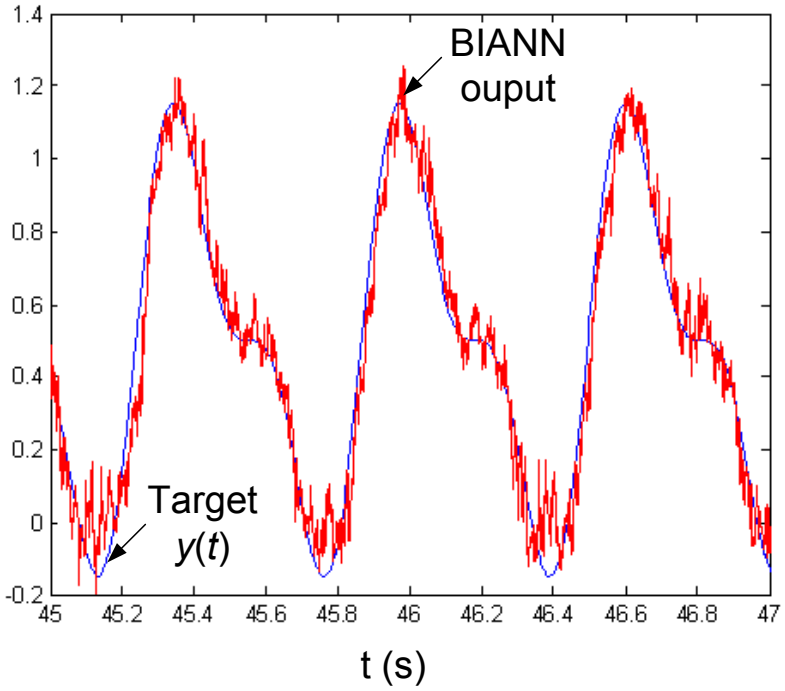


Figure 6.10: Modeling results of the BIANN without output feedback.

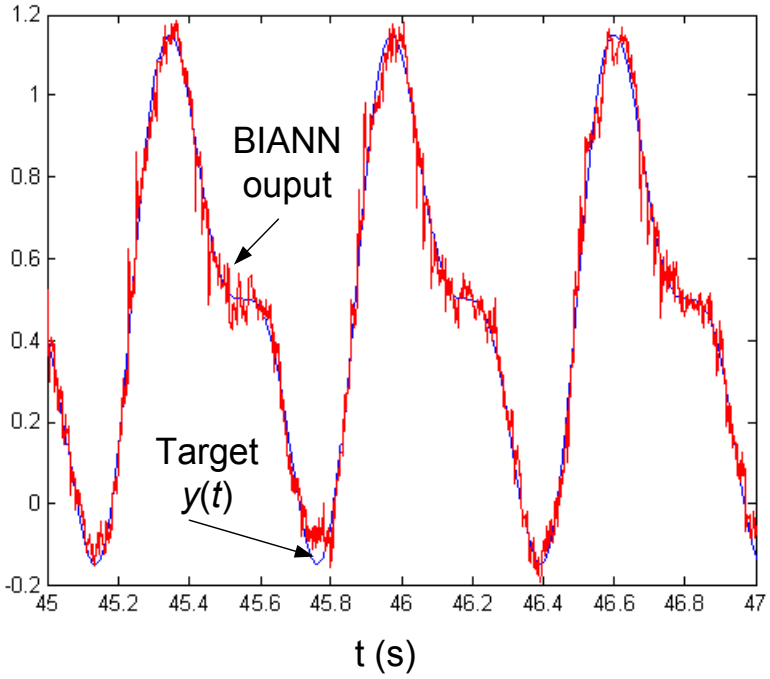
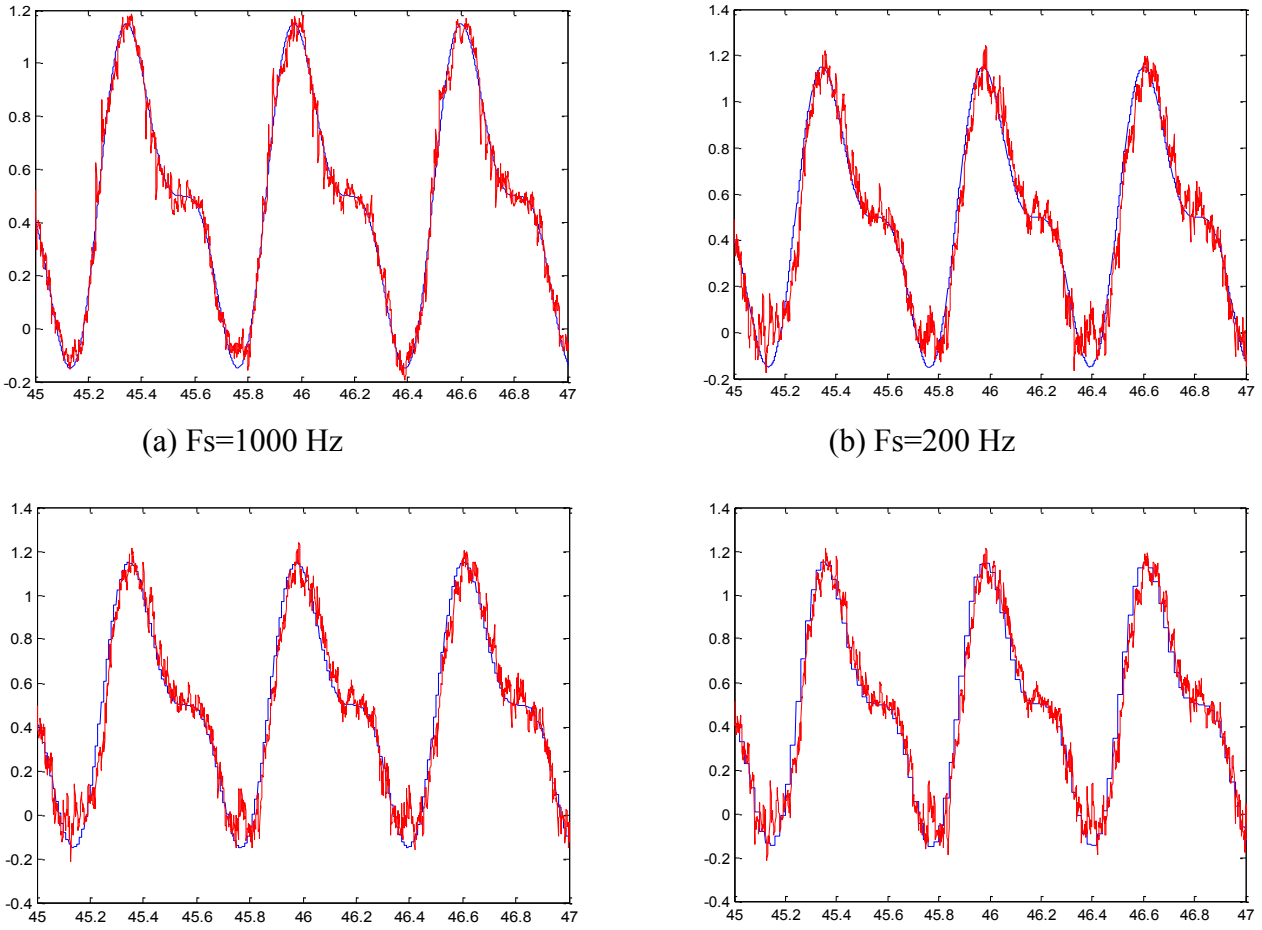


Figure 6.11: Modeling results of the BIANN with output feedback.

6.4.6 Impact of Limited Sampling Rate

In practice, the modeling of dynamic systems is usually limited by the sampling rate of the input and output signals collected. For the BIANN, the time step for modeling or calculation is always set to be 1 ms, due to its biological meaning. The impact of different sampling rates is illustrated in Figure 6.12. Four different conditions: sampling rate of 1 kHz, 200 Hz, 100 Hz and 50 Hz are compared. The MSEs for these 4 conditions are 0.0024, 0.0018, 0.0021 and 0.0027, respectively. Therefore, the modeling capability of the system is not largely affected by the limited sampling rate for this specific example. However, for practical applications, the selection of sampling rate depends on the dynamics of the plant.



(a) Fs=1000 Hz

(b) Fs=200 Hz

(c) Fs=100 Hz

(d) Fs=50 Hz

Figure 6.12: Modeling results of BIANN with output feedback using different input sampling frequencies.

6.5 Chapter Summary

As an effort to develop the next-generation artificial intelligence, this Chapter focuses on the development of Biologically Inspired Artificial Neural Networks (BIANNs). Unlike traditional artificial neural networks, more brain-like spiking neuron models are proposed to better simulate living neural networks.

The structure of a BIANN is proposed with spiking neural network as the core component. The encoding and decoding approaches are both proposed to incorporate

such spiking neural networks for online modeling of dynamic systems. A reservoir-computing-based training algorithm is adopted in the BIANN for reducing the required computational effort. Training approaches for the proposed BIANN have been presented for forced learning and training of the BIANN. During online training, only the output weights are updated at each time step, which guarantees computational effectiveness of the BIANN. Simulation results have been presented to illustrate the proposed BIANN and its training approaches.

The BIANN can be potentially extended to real living neural networks. It is possible to replace spiking neuron models with living neural networks to realize training using living neurons for modeling or control of complex systems.

The effectiveness of the proposed BIANN is demonstrated in an example in this Chapter. However, the performance of the BIANN is not validated in practical applications. The usefulness of the BIANN for modeling, monitoring and control applications in power system will be evaluated in Chapters 7 and 8.

CHAPTER 7 BIANN FOR MODELING OF A SINGLE MACHINE INFINITE BUS POWER SYSTEM

7.1 Overview

The BIANN, which is based on spiking neuron models of LNNs, processes information in a more "brain-like" fashion than conventional ANNs. A reservoir-computing-based training approach is proposed in this chapter for BIANNs to serve as a novel modeling and control tool for practical applications. The feasibility of the proposed BIANN is illustrated for the prediction of a synchronous generator's speed and terminal voltage signals in a single machine infinite bus (SMIB) electric power system. The proposed BIANN model is able to provide an accurate prediction for online identification of a generator.

7.2 Online Identifying of the SMIB System

7.2.1 The SMIB System

The infinite bus in Fig. 7.1 is consists of an ideal voltage source, called an infinite bus, and a network that is represented by algebraic equations, since ignoring the network dynamics is common practice in transient stability studies [119]; therefore, the only dynamics in the system are those of the generator, its exciter and automatic voltage regulator (AVR), and its turbine and governor.

Generator rotor speed and terminal voltage are critical quantities to monitor and control. The time-ahead predictions of these quantities are required by the controllers of rotor speed and terminal voltage which indirectly affect the real and reactive power produced by the generator.

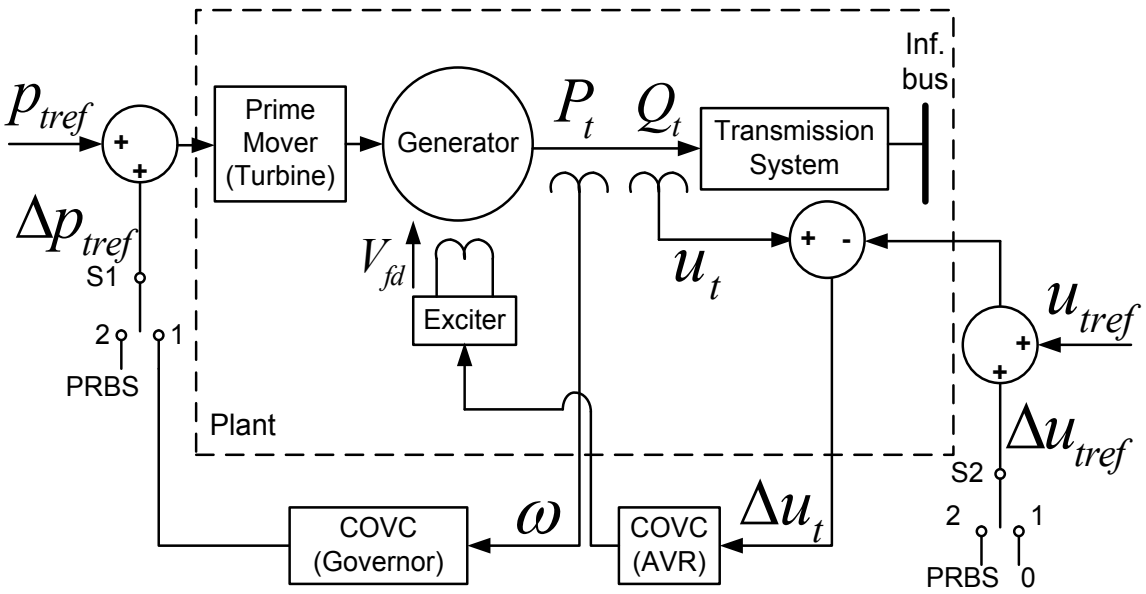


Figure 7.1: The single machine infinite bus (SMIB) system.

7.2.2 Online Training and Testing Scheme of the BIANN Model for the SMIB System

The goal of the BIANN model is to learn the dynamics of the SMIB system using only input-output data collected a-priori to produce predictions of the terminal voltage and rotor speed. The online k-step-ahead prediction scheme is shown in Fig. 7.2. The SMIB system dynamics are described by (7.1):

$$\mathrm{y(n+1)} = \mathrm{f\big(u(n), y(n)\big)} \tag{7.1}$$

where $u(n) = [u_{t_{ref}}(n) + \Delta u_{t_{ref}}(n),\ p_{t_{ref}}(n) + \Delta\, p_{t_{ref}}(n)]$ and

$y(n) = [\omega(n),\ u_t(n)]$.

ω and u_t are the rotor speed and terminal voltage of the generator respectively. $u_{t_{ref}}$ and $p_{t_{ref}}$ are the terminal voltage and active output power references to the AVR and turbine governor, respectively.

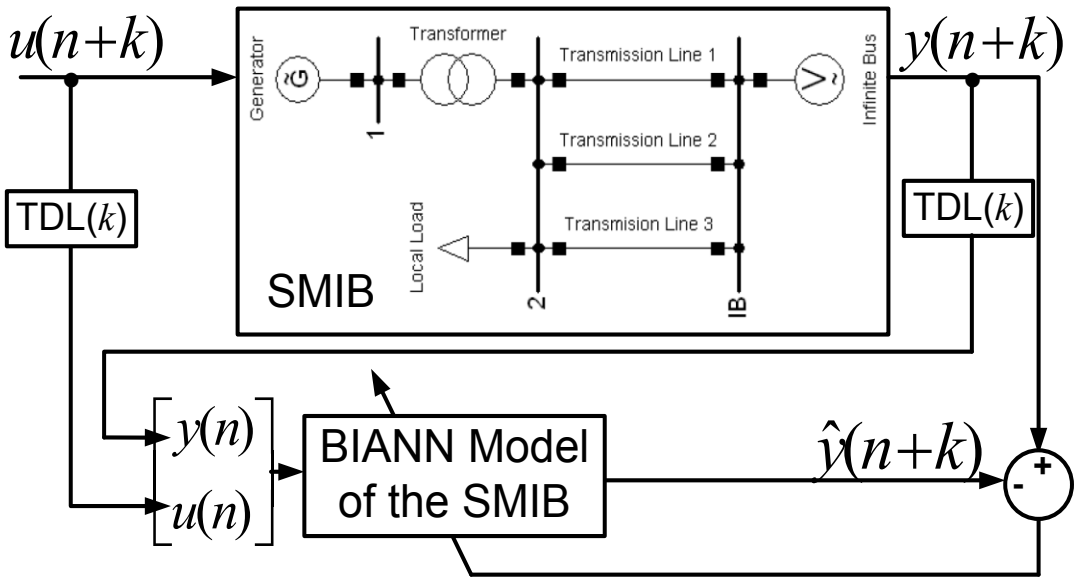


Figure 7.2: Online generator responses prediction scheme.

For this application, each sample is taken at 10 ms intervals. At each sampling step, the BIANN model is updated based on the previous inputs and outputs of the system and provides *k*-step-ahead prediction of the system outputs.

7.2.3 Training Data

The details of encoding, decoding, online training and testing algorithms of the BIANN model for the SMIB system are given in this section. The steady state values of u_{tref} and p_{tref} are 1.025 p.u.and 0.714 p.u. respectively.

The SMIB system is simulated using PSCAD, which is a commercial power system simulation software package. Pseudo-Random Binary Signals (PRBS) Δu_{tref} and Δp_{tref} are added to the terminal voltage and the active power set point respectively, so that *u(n+k)* at the input to Fig. 7.3 is equal to $[(1.025+\Delta u_{tref}),(0.714+\Delta p_{tref})]$.

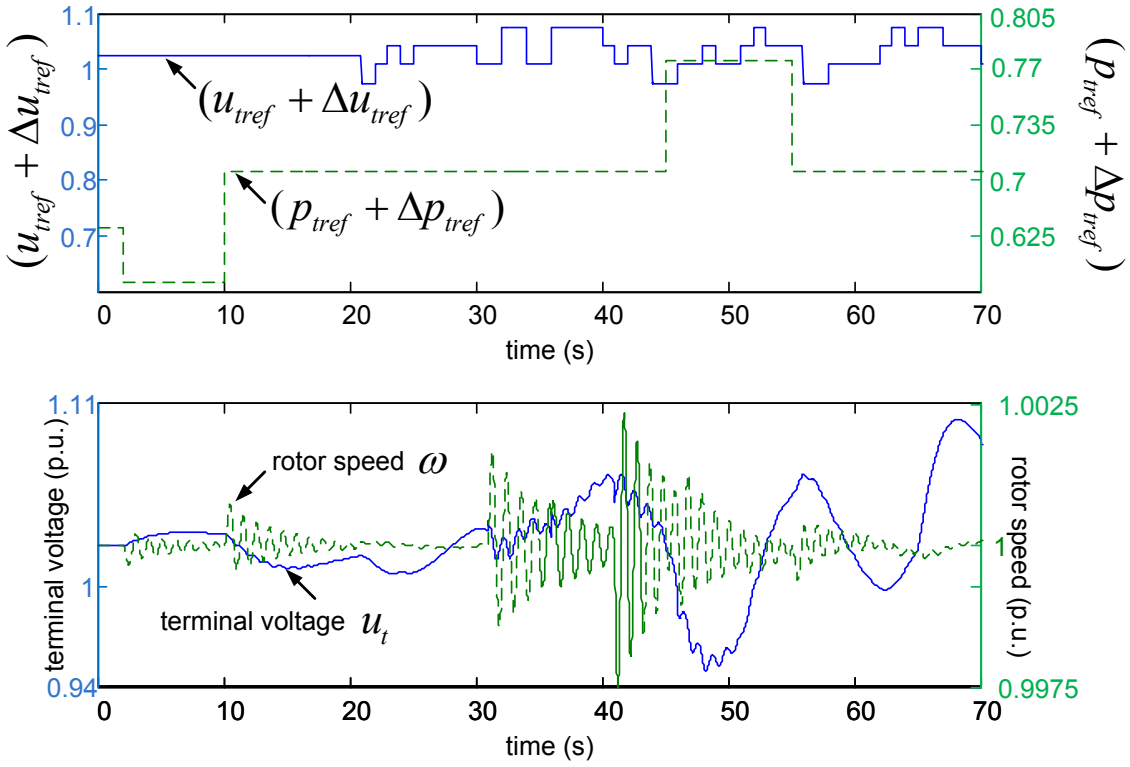


Figure 7.3: Training data: input with PRBS and output of the SMIB system.

This input is plotted in the top part of Fig. 7.3. The PRBS excites the dynamics of the SMIB system. The rotor speed ω and terminal voltage u_t of the generator exhibit small oscillations around their steady state values, as shown in the bottom part of Fig. 7.3. The input and output data of the SMIB block in Fig.7.2 are then sampled from Fig. 7.3 as the training data set of the BIANN. The sampling frequency of the training data is 100 Hz, or every 10 ms.

7.2.4 Encoding Process

At each time step *n*, the sampled input vector to the BIANN model of Fig. 7.2 has four variables, as shown in (7.2):

$$input_{1-4}(n) = [u(n), y(n)]$$

$$=[u_{t_{ref}}(n) + \Delta u_{t_{ref}}(n),\ p_{t_{ref}}(n) + \Delta\, p_{t_{ref}}(n), \omega(n),\ u_t(n)], \tag{7.2}$$

The goal of the encoding process is to convert these continuous signals into spike trains, which is the form required to stimulate the spiking neurons in the BIANN. Each of the four variables is encoded into *M* parallel spike trains (here, *M*=10 is used), hence after the encoding process, the four continuous signals of Fig. 7.3 become forty spike sequences.

The same mean-firing-rate based encoding algorithm as described in Section 6.3.1 is used to convert the training data into spike trains. This mean-firing-rate-based encoding method is reversible, in other words, the original signal can be accurately recovered from the spikes using a reverse algorithm. As an example, the encoding result of the third element in the input vector in (7.2), which is the rotor speed of the generator $\omega(n)$, is shown in Fig. 7.4, and Fig. 7.5 shows that the original signal can be recovered from these spikes.

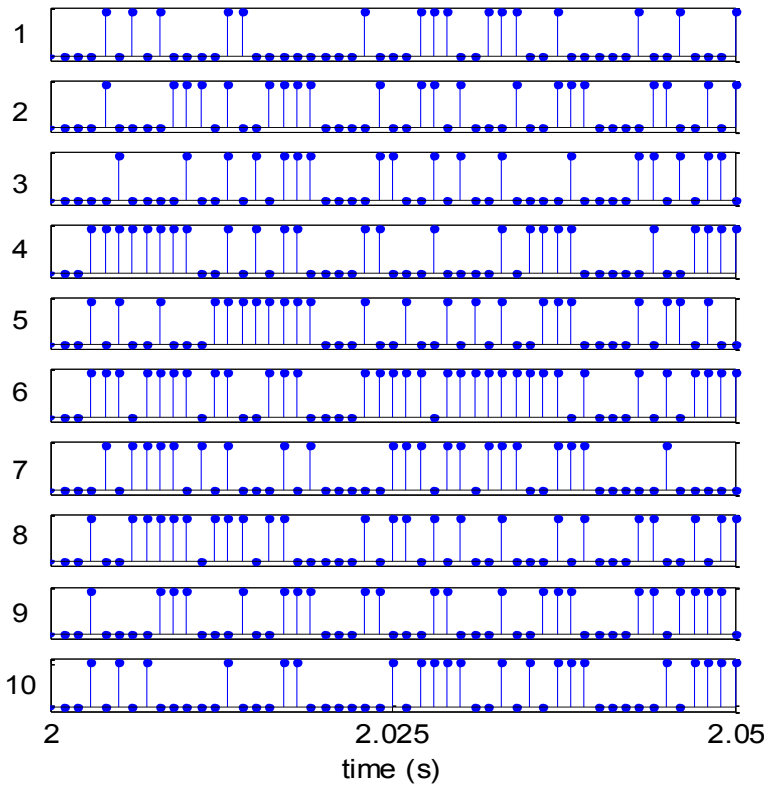


Figure 7.4: One of the encoded inputs: the rotor speed.

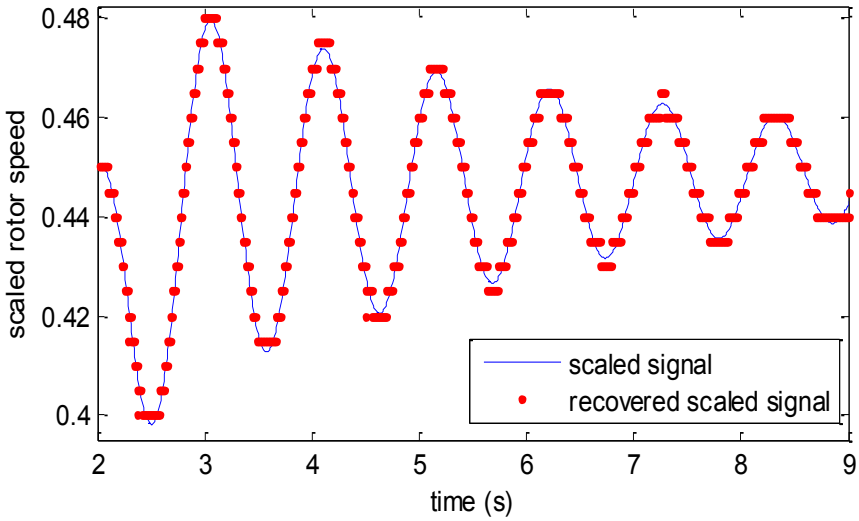


Figure 7.5: Reversibility of the encoding algorithm to recover signal from spike data in Fig. 7.4.

7.2.5 Decoding Process

The computation in the dynamic reservoir strictly follows the Izhikevich spiking neuron model described Section 6.2.2 and 6.2.3. When a neuron fires at time step *n*, the output of this neuron at this moment is marked as $Fire_i\,(n)$=1, otherwise, $Fire_i\,(n)$=0.

The decoding method is the well known mean firing rate method [84]. At time step *n*, the output of the i^{th} neuron in the BIANN is decoded from spiking signals $Fire_i$ to continuous signals using (7.3):

$$output_i(n) = \frac{1}{T}\sum_{\tau=n-T+1}^{n} Fire_i\,(\tau) \tag{7.3}$$

T=20 ms is the averaging window. $output_i(n)$ is not the final output of the BIANN, since an additional readout layer with trainable weights is still needed to extract useful information from the dynamic reservoir.

7.2.6 Training Algorithm of the Readout Weights

Based on the algorithm proposed in [67], the training and testing of the BIANN model for the SMIB system consists of the following steps:

- Use a matured BIANN as the initial network of the online training process;

How to mature a BIANN has been described in Section 6.4.3 in chapter 6. Start feeding the encoded input data into the BIANN, and let the input evoke rich dynamics in the dynamic reservoir. The internal weights in the dynamic reservoir are adjusted following the STDP rule described in Section 6.2.3. This is different from the original Echo State Network (ESN) [67] and Liquid State Machine (LSM) [68] theories. In both ESN and LSM theories, the internal weight matrix of the dynamic reservoir or the liquid is fixed as soon as it has been generated. It is believed that the input history is memorized and stored in the states (values) of each internal neuron in the dynamic reservoir.

However, in the case of the BIANN proposed in this chapter, the internal dynamic reservoir is always changing according to the STDP rule. The "memory" of the BIANN is affected by two factors: the states of each neuron and the weights in both the dynamic reservoir and the readout layer.

- Collect 10 seconds of decoded output from the dynamic reservoir and save it in a matrix called X(n). $X(n) = [output(1), output(2), \dots, output(n)]$ is a 1000-by-10000-dimensional matrix since all 1000 neurons (including 40 input neurons) in the dynamic reservoir are connected to the final output of the BIANN via the readout layer and the Izhikevich model runs at a frequency of 1000 Hz;

k steps later from the current moment n, when the actual output $y(n+k)$ from the SMIB system becomes available, also put 10 seconds of y in a matrix called $Teacher(n)$. $Teacher(n) = [y(1+k), y(2+k), \dots, y(n+k)]$ is a 2-by-10000-dimensional matrix since there are two variables that need to be predicted;

- Calculate the readout weight matrix $W^{out}(n)$ using a psuedoinverse algorithm as in (7.4);

$$W^{out}(n) = Teacher(n) \cdot psuedoinverse(X(n)) \qquad (7.4)$$

In this thesis, $W^{out}(n)$ has a dimension of 2-by-1000.

Given a teacher signal, the training of the BIANN is realized by updating the readout weights. Meanwhile, the weights in the dynamic reservoir are spontaneously updated based on the STDP rules described in section 6.2.3. The two types of updates are of different nature. The STDP update simulates the biological process in the brain that adjusts the strength of connections between neurons. The readout weight updates belong to the category of supervised learning. The goal of the training of the readout weights is

to build the relation between rich patterns produced by the dynamic reservoir and the teacher outputs.

7.2.7 Prediction using Trained BIANN

After the five steps described in the section above, the BIANN is trained and ready for use. The weights in the readout layer are now fixed, but the weights in the dynamic reservoir are still updating according to the STDP rule.

To capture any possible system change for modeling purposes, a moving window technique is used during the training and prediction of the BIANN. As shown in Fig.7.6, the network is first trained with 10 seconds of data and then used for prediction for the following 5 seconds. The window shifts 5 seconds to the right after a complete cycle of training and prediction has been completed.

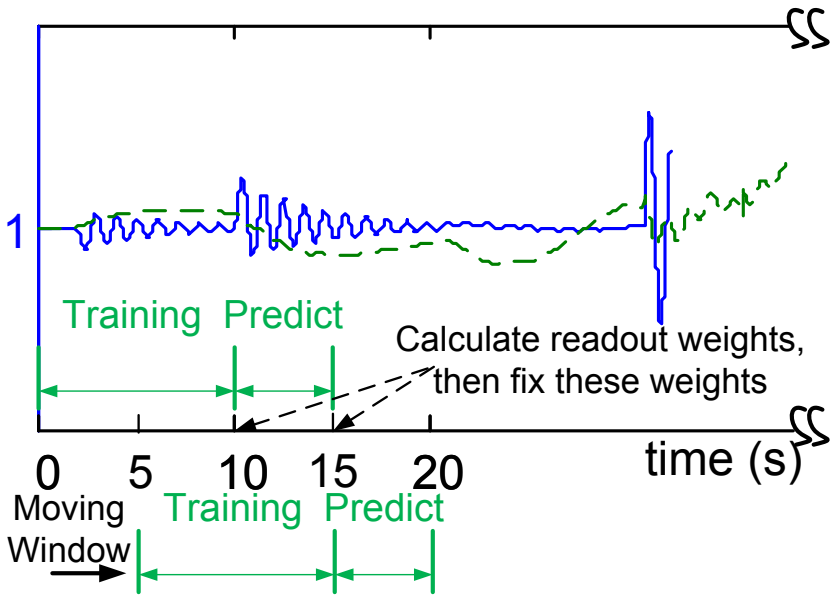


Figure 7.6: Moving window training and prediction scheme.

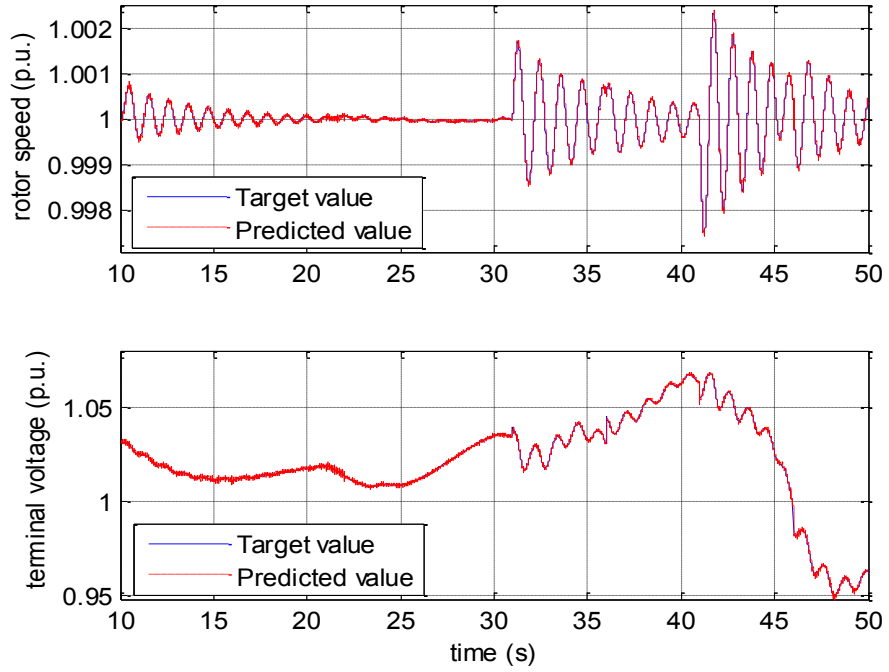
The final output of the BIANN, i.e. the *k*-step-ahead predicted rotor speed and terminal voltage of the generator are calculated using (7.5):

$$\hat{y}(n+k) = [\hat{\omega}(n+k), \hat{u}_t(n+k)] = W^{out}(n) \cdot output_i(n) \qquad (7.5)$$

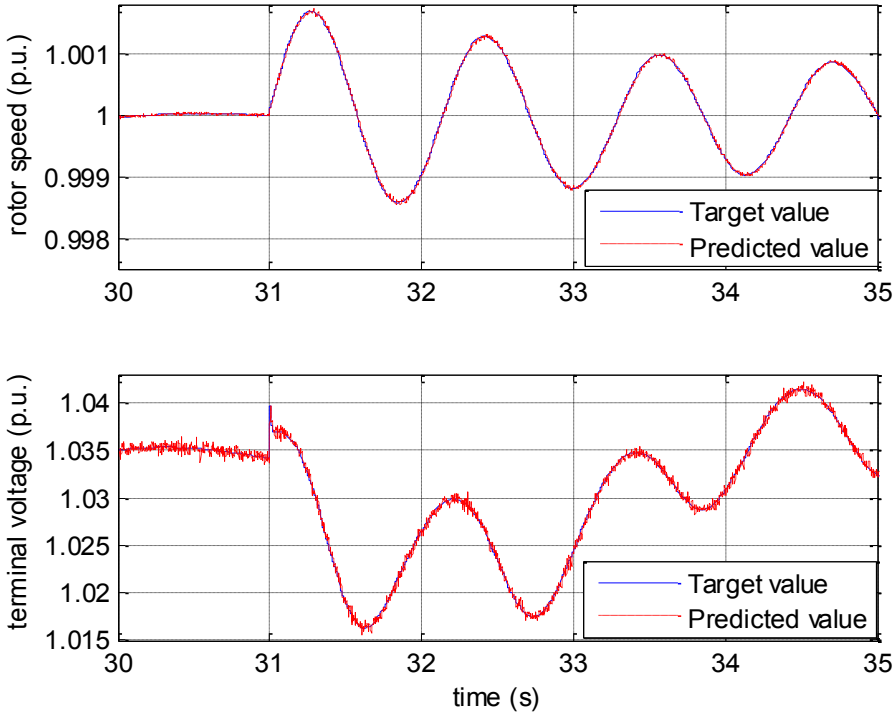
7.3 Results and Discussion

7.3.1 One-Step Ahead Prediction Results

A BIANN model is first trained to predict the generator speed and terminal voltage under the operating conditions described in Section 7.2.3. The BIANN model is trained using the moving-window approach as described in Section IV.E. The online prediction results of the BIANN model are shown in Fig. 7.7. Fig. 7.7(a) shows the prediction performance over a 40 second time span while Fig. 7.7(b) shows the prediction results over a 5 second time span. These results show that the BIANN model can accurately predict both rotor speed and generator terminal voltage.



(a)

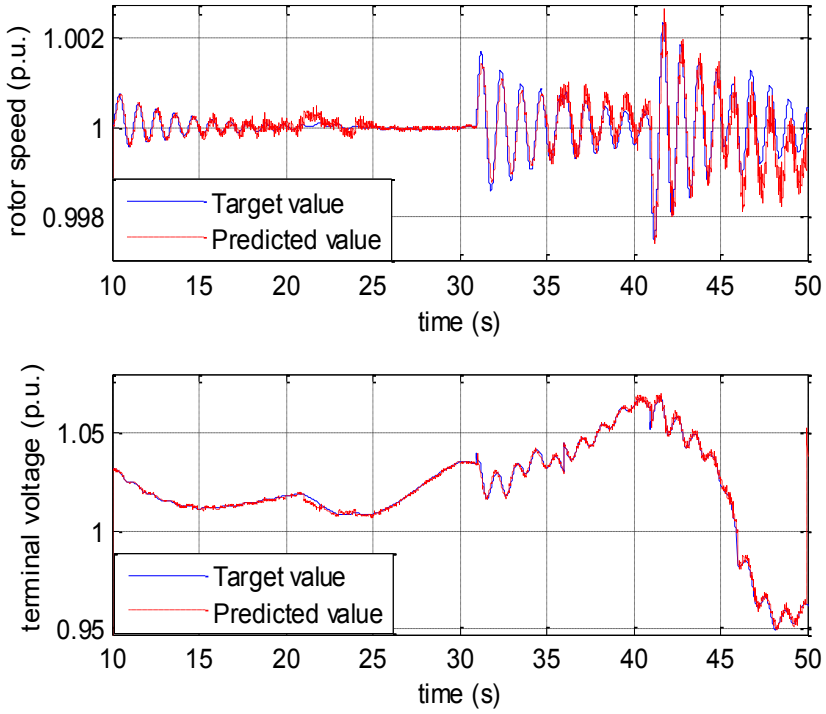


(b)

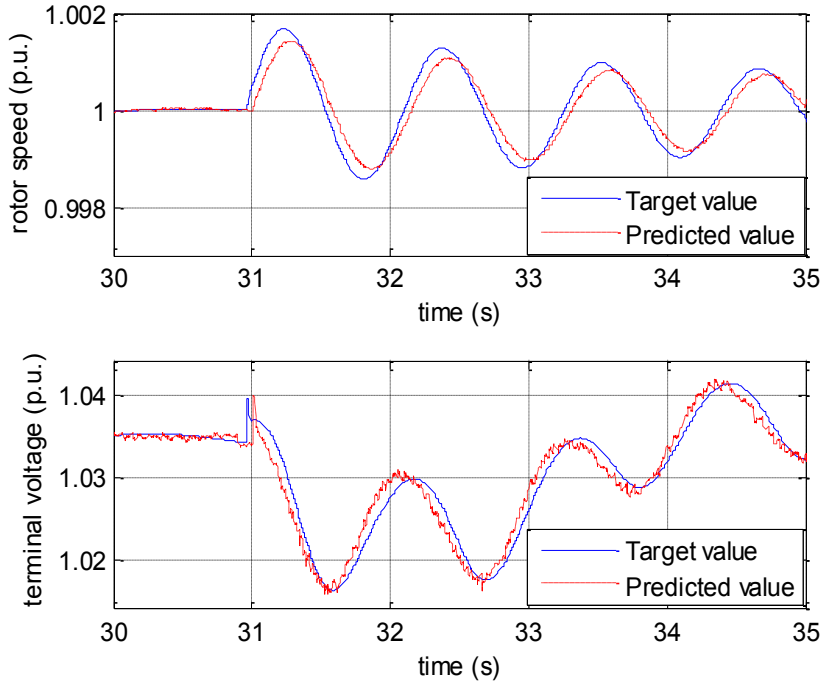
Figure 7.7: One-step-ahead prediction of generator speed and terminal voltage using BIANN model.

7.3.2 Five-Step Ahead Prediction Results

As discussed in Section 7.2.1, it is not only important to accurately predict the generator rotor speed and the terminal voltage, but also critical to predict these quantities in a multi-step-ahead fashion so that these predictions can more effectively assist the monitoring and control of the power system. Therefore, a similar exercise is conducted for the BIANN model to provide five-steps-ahead prediction of the generator rotor speed and terminal voltage. The five-steps-ahead prediction performance of the BIANN is illustrated in Fig. 7.8. The comparison between Fig. 7.7 and Fig. 7.8 shows that the five-steps-ahead predictions are less accurate than the one-step-ahead predictions, which is expected since less relevant information is used for the five-steps-ahead prediction.



(a)



(b)

Figure 7.8: Five-steps-ahead prediction of generator speed and terminal voltage using BIANN model.

7.3.3 Prediction Accuracy versus Number of Steps Ahead

The mean absolute relative errors (MAREs) between 10 and 50 seconds of the generator rotor speed and terminal voltage predictions, using BIANN models in five-steps-ahead fashion, are defined by (7.6) and compared in Fig. 7.9.

$$MARE = \frac{1}{40\times100}\sum_{n=1000}^{5000}\left|\frac{y(n)-\hat{y}(n)}{y(n)}\right| \times 100\% \qquad (7.6)$$

As expected the prediction error increases with more-step-ahead predictions. However, in practical applications, the selection of the prediction steps is an important tradeoff between prediction accuracy and the time allowed by the prediction for predictive operation and control in the power system. This tradeoff selection is even more important for wide-area monitoring and control of large-scale power systems due to the communication and operational delays.

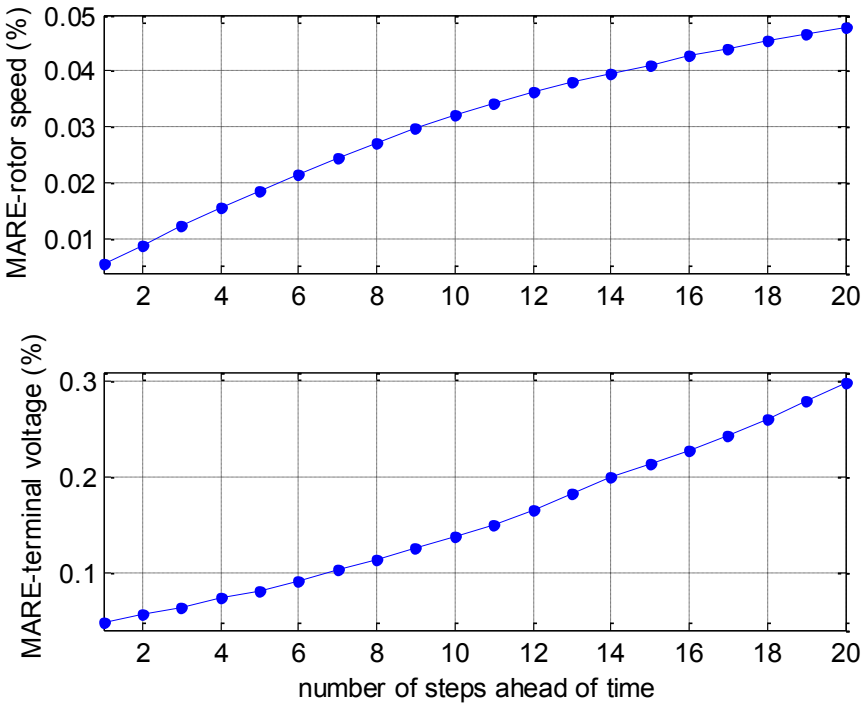


Figure 7.9: MAREs of generator speed and terminal voltage predictions using BIANN models versus prediction steps.

7.4 Chapter Summary

Artificial neural networks have been used in various applications as a computational intelligence tool. Although inspired by living neuronal network in the brain, ANNs have many fundamental differences compared to LNNs, such as information transmission format, neuronal plasticity. Due to these fundamental differences, the training mechanisms for ANNs cannot be extended to LNNs. As a result, it has been challenging to use LNNs as computational intelligence tools.

LNNs and their mathematical models, e.g., spiking neuron models, are broadly believed to have huge computational capability. To bridge the gap between ANNs and LNNs, a novel type of artificial neural network, i.e. biologically-inspired artificial neural network (BIANN) is proposed in this Chapter. The BIANN, which is based on the spiking neuron models of LNNs, processes information in a more "brain-like" fashion than conventional ANNs. A reservoir-computing-based training approach is also proposed for the BIANN to serve as a novel modeling and control tool for practical applications.

The generator rotor speed and terminal voltage are critical quantities for monitoring and operating power systems. The BIANN provides one-step-ahead and five-step-ahead predictions of these quantities for online monitoring of the generator in a single machine infinite bus power system. The prediction accuracy degrades as the number of steps-ahead increases.

Accurately modeling the dynamics of the SMIB system is an important preparation for the next step, which is to develop an adaptive-critic-design (ACD)-based optimal control scheme for the generator in the SMIB system. The implementation of the ACD-

based control scheme for the generator is described in detail in Chapter 8. This is the first time that a BIANN is used in a close-loop control application.

CHAPTER 8 ADAPTIVE-CRITIC-BASED OPTIMAL CONTROL FOR A TURBO GENERATOR IN A SINGLE MACHINE INFINITE BUS POWER SYSTEM USING BIANNS

8.1 Introduction

The BIANN has been successfully used to model or identify the dynamics of a SMIB power system in Chapter 7. However, the feasibility of using a BIANN in closed-loop neurocontrol still needs to be studied. The turbo generator in the SMIB power system is a nonlinear, fast-acting, multiple-input-output device. Conventional linear controllers (CONVCs) for the generator consist of the automatic voltage regulator (AVR) to maintain constant terminal voltage and the turbine governor to maintain constant speed at some set point. CONVCs are usually designed around one particular operating point, so that when the operation point changes or a disturbance happens, the controllers' performances are degraded. ANNs offers an alternative for the CONVC as nonlinear adaptive controllers. Researchers have used different types of neural networks, for example, the MLPNN and the RBFNN in single and multi-machine power system studies [120]. It is important to show that the newly developed BIANN can also be used in such applications. Moreover, a comparison between the traditional neural networks and the brain-like, spiking neural networks is given for a specific power system control application.

In this chapter, BIANNs will be used to control a turbo generator in a SMIB power system within the adaptive critic design (ACD) framework. First, the background of ACD with relation to optimal control theory is given. Secondly, an optimal neurocontroller using BIANNs based on the heuristic dynamic programming (HDP), which is a class of

the ACD family is designed for the turbo generator in the SMIB system. The performances of the BIANN controller are shown in two case studies. Then a comparison is made between the BIANN and a traditional neural network, i.e. the MLP, to control the generator.

8.2 Background on Adaptive Critic Designs

This section introduces the foundations of adaptive critic designs (ACDs) [121]. The ACD technique for optimal control problems was developed in the 1990s. In optimal control problems, the objective is to obtain a trajectory of actions, or control law that optimizes a desired performance metric or cost. ACDs utilize two parametric structures called the actor and the critic. The actor (or controller) consists of a parameterized control law, and the critic approximates a value-related function and captures the effects that the control law will have on the future cost. At any given time, the critic provides guidance on how to improve the control law. An algorithm that successively iterates between the actor and the critic eventually converges to the optimal solution.

8.2.1 The Optimal Control Problem

Consider a continuous-time dynamic system described by (8.1):

$$\dot{x}(t) = f(x(t), u(t), t), \qquad 0 \le t \le T \tag{8.1}$$

where $x(t) \in \mathrm{R}^{\mathrm{n}}$ is the state vector at time t, $\dot{x}(t) \in \mathrm{R}^{\mathrm{n}}$ is the vector of first order time derivatives of the state vector at time t, $u(t) \in \overline{\mathrm{U}} \in \mathrm{R}^{\mathrm{m}}$ is the control vector at time t, $\overline{\mathrm{U}}$ is the control constraint set, and T is the terminal time.

The cost-to-go function[121] at time t and state x is defined in (8.2).

$$J(t,x) = h\big(x(T)\big) + \int_0^T g(x(t),u(t))\,dt \tag{8.2}$$

where h is the cost associated with the error in the terminal state at time T, and g is the cost function associated with transient state errors and control effort.

The optimal control problem can then be considered as finding $u \in \overline{\mathrm{U}}$ to minimize the total cost-to-go function J subject to the dynamic system constraints in (8.1) and all initial and terminal boundary conditions.

8.2.2 Adaptive Critic Designs (ACDs)

The continuous-time cost function J in (8.2) can be reformulated as the total cost-to-go function of the infinite horizon problem described in (8.3) for a discrete-time dynamic system.

$$J_\pi(x_0) = \sum_{k=0}^{\infty} \gamma^k g(x(k),u(k)) \tag{8.3}$$

where $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a control policy $\pi = \{u_0,\ u_1, \ldots\}$, and γ is the discount factor ($0 < \gamma < 1$).

The Bellman equation [121] using dynamic programming (DP) can be formulated in (8.4), which is solved iteratively at each time step to find the optimal control policy u^* corresponding to the optimal cost-to-go function J^*, in (8.5).

$$J_{k+1}(x) = \min_{u\in\bar{U}}\big[g(x,u) + \gamma J_k\big(f(x,u)\big)\big], \quad k = 0,1,\ldots$$

$$J_0(x) = 0, \ \text{for all } x \tag{8.4}$$

$$J^*(x) = \lim_{k\to\infty}(T^k J_0)(x), \ \text{for all } x \in S \tag{8.5}$$

where $(TJ)(x)$ is a DP mapping function defined in (8.6) on the state space S for any function J: $S \to \mathbf{R}$

$$(TJ)(x) = \min_{u\in\bar{U}}[g(x,u) + \gamma J(f(x,u))]. \tag{8.6}$$

Since the classical DP algorithm requires extensive computations and memory, known as "the curse of dimensionality", the optimal control theory mentioned above cannot be directly applied to deal with practical nonlinear control problems. Several alternative approaches have been proposed to overcome this problem. These methods can approximate the cost-to-go function. Adaptive critic design (ACD) technique, which uses function approximators, e.g. neural networks, is one of these alternative methods. The ACD technique combines the concept of reinforcement learning and approximate dynamic programming. ACDs are based on an algorithm that cycles between a "policy-improvement routine" and a "value-determination operation". There are four categories of adaptive critic designs: heuristic dynamic programming (HDP), dual heuristic dynamic programming (DHP), globalized dual heuristic dynamic programming (GDHP) and action-dependent (AD) design [118]. They are all based on the adaptive critic algorithm, but differ in what functional they set out to approximate.

8.2.3 Heuristic Dynamic Programming (HDP)

Heuristic dynamic programming (HDP), which uses the parametric structure called the "actor" to approximate the control law, and another parametric structure called the "critic" to approximate the value function, is a class of the ACD family. The structure of the HDP configuration is shown in Fig. 8.1. TDL is a time delay by one time step. In practice, this represents the operating cycle of the controller. The derivative of the cost function $J^{\mathrm{c}}(k)$ with respect to the control command $\mathrm{A}(k)$ is obtained by backpropagating $\partial J^{\mathrm{c}}(k)/\partial J^{\mathrm{c}}(k)$ (the number "1" shown in Fig. 8.1) through the critic network and then through the model network to the action network. The propagation through the critic network represents the computation of the desired change of the plant

response predicted by the model network to minimize the cost-to-go function *J*; the propagation through the model network computes the desired adjustment of the action network output, which is also the input to the plant. The action network is then adjusted so that the desired control signal is given to the plant to minimize the cost-to-go function. When all three networks converge at the same time, the cost-to-go function is minimized and the actual control signal is optimized.

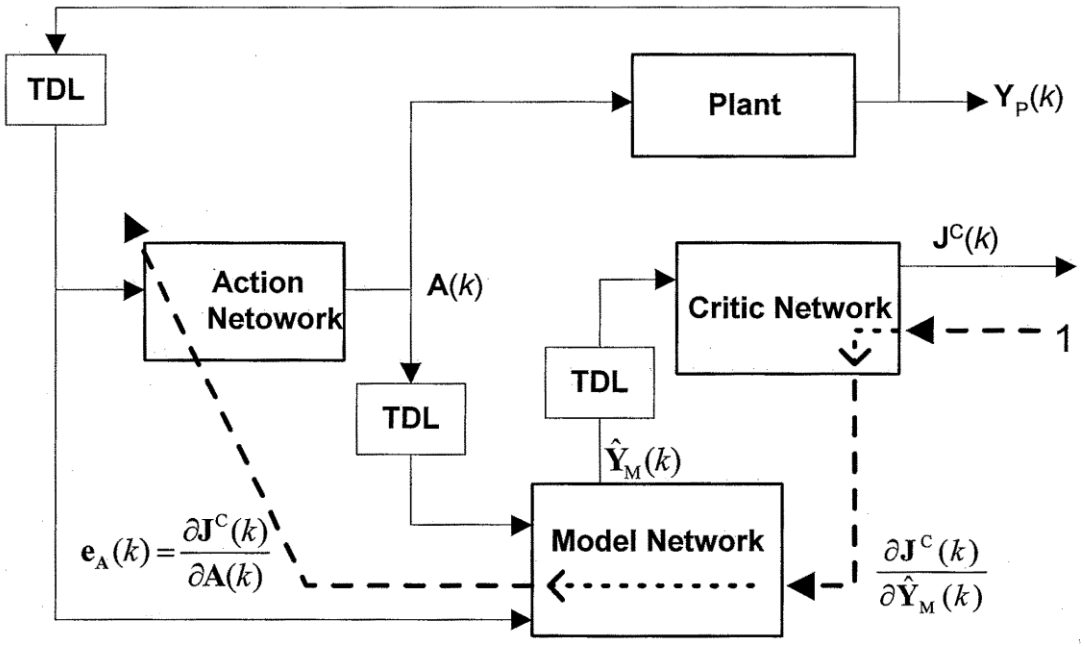


Figure 8.1: Structure of the HDP configuration: action adaptation in HDP.

HDP uses three different neural networks, namely, the critic network, the action network and the model network. In HDP, the utility function (or cost function) $U^{\mathbf{c}}$ to be minimized is called "reinforcement". There are two major steps to account for the link between present actions and future consequences. The first step is to build a "model" network for identifying the plant, and use backpropagation through the model network to

calculate the derivatives of the future cost with respect to present actions through the model network. The second step is to adapt a "critic" network, which outputs an estimate of the total future value of $U^{\mathbf{c}}$. In other words, the critic network learns to approximate the heuristic cost-to-go function in (8.7).

$$J^{\mathrm{c}}(k) = \sum_{p=0}^{\infty} \gamma^{p} U^{c}(k+p) \tag{8.7}$$

where γ is the discount factor and $0< \gamma < 1$.

After minimizing the cost-to-go function J^{c} by the critic network, the action network is trained with the estimated output backpropagated from the critic network to obtain the converged weight for the optimal control law u^{*}. The detailed approach is presented in the following section.

The design and training of the model, critic and action network are described in Section 8.3 together with their mathematical analyses.

8.3 HDP-based Control for a Turbo Generator in a SMIB Power System using BIANNs

8.3.1 The Plant

The plant that needs to be controlled consists of a turbo generator, a transmission line and an infinite bus. As shown in Fig. 8.2, in the plant,

P_t and Q_t are the real and reactive power at the generator terminal in p.u.,

P_m is the mechanical input power to the generator in p.u.,

V_{fd} is the field voltage of the exciter in p.u.,

V_b is the voltage at the infinite bus in p.u.,

$\Delta\omega$ is the speed deviation in p.u.,

ΔV_t is the terminal voltage deviation in p.u.,

V_t is the terminal voltage in p.u.,

ΔV_{ref} is the reference voltage deviation in p.u.,

V_{ref} is the reference voltage in p.u.,

ΔP_{in} is the input power deviation in p.u. and

P_{in} is the input power of the turbine in p.u..

The positions 1 and 2 of switches S1 and S2 in Fig. 8.2 determine whether the HDP-based BIANN controller, or the conventional controller (CONVC) which consists of a governor and an AVR, is controlling the plant.

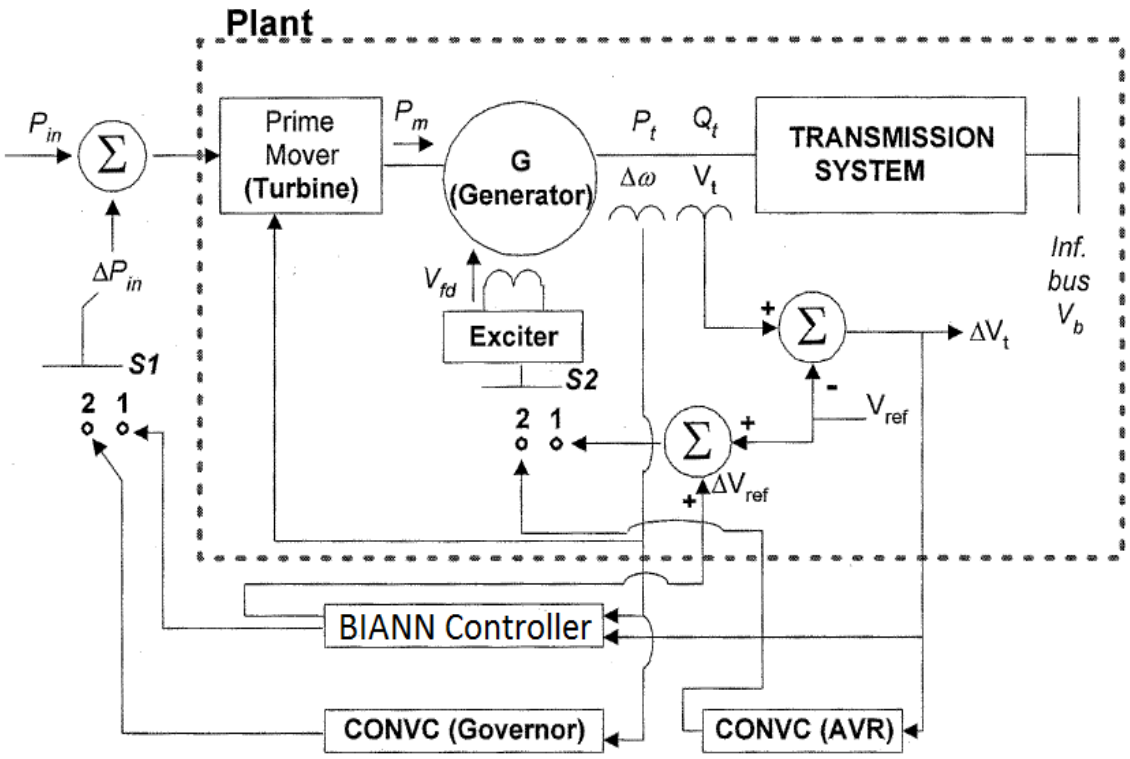


Figure 8.2: The plant: a single machine infinite bus power system.

8.3.2 Design and Training of the Model Network

How the model network (identifier) is trained online to identify the dynamics of the plant has been described in Chapter 7 in detail.

At this stage in the training, there is no action network or critic network or CONVC present. The training is carried out at several different operating conditions within the stability limit of the generator until the testing error of the model network has converged to a small value. See Chapter 7 for the detailed modeling technique using BIANNs.

8.3.3 Design and Training of the Critic Network

The critic network in the HDP approximates the function J^{c} in (8.7). The configuration for training the critic network is shown in Fig. 8.3. The Bellman equation in DP in (8.4) is implemented by the ADP using two critic networks.

The initial cost-to-go function J^{c} at time zero has a positive value, because the initial weights $W_{\mathrm{C}}(0)$ of the critic network are randomly generated and the value of J^{c} keeps minimizing as the time goes to infinite. Therefore, the error equation (8.8) for the adaptation of the critic network can be obtained as follows [120].

$$e_C(k) = J^c(k) - \gamma J^c(k+1) - U^c(k) \tag{8.8}$$

where $J^c(k)$ and $J^c(k+1)$ are the outputs of the critic network for two consecutive times k and k+1, $U^c(k)$ is the utility function that is defined in (8.9), and a discount factor $\gamma = 0.8$ is used here. The value of the discount factor should be within the range of (0, 1) to guarantee the convergence of the cost-to-go function. Recall equation (8.7), the cost-to-go function J^{c} is the sum of all utility functions from the current moment to the infinite time-horizon. The discount factor represents the weight of the current cost in comparison to the futuristic costs. A small discount factor indicates that the cost-to-go function

emphasizes more on the near-future cost; while a large discount factor indicates the cost-to-go function emphasizes more on the longer-term cost. In this application, an discount factor of 0.8 is used.

The design of the utility function is based on several trial runs, and the choice of the coefficient in front of each term is related to how important this term is in determining controller performance.

$$U^c(k) = 4\Delta V_t(k)^2 + 4\Delta V_t(k-1)^2 + 1.6\Delta V_t(k-2)^2 + 0.4\Delta\omega(k)^2 + 0.4\Delta\omega(k-1)^2 + 0.16\Delta\omega(k-2)^2 \quad (8.9)$$

The critic network's weights, more specifically, the readout weights W_C are updated using (8.10) and (8.11) [117] to minimize the mean square error $\frac{1}{2}e^2{}_C(k)$.

$$W_C(k+1) = W_C(k) + \Delta W_C(k) \quad (8.10)$$

$$\Delta W_C(k) = -\eta_C \cdot \frac{\partial\left[\frac{1}{2}e^2{}_C(k)\right]}{\partial W_C(k)} = -\eta_C \cdot e_C(k) \cdot \frac{\partial e_C(k)}{\partial W_C(k)} \quad (8.11)$$

where η_C is a positive learning rate.

The training for the critic network by the backpropagation algorithm is carried out until the value of J^c has converged to as small a value as possible, which is almost zero. The adaptation process eventually reaches the optimal cost-to-go function *J**. *J** is the optimized cost-to-go function value, which in theory is the minimal cost from the current moment to the infinite horizon.

The configuration for training the critic network is shown in Fig. 8.3 using outputs from the critic network at two successive time instants.

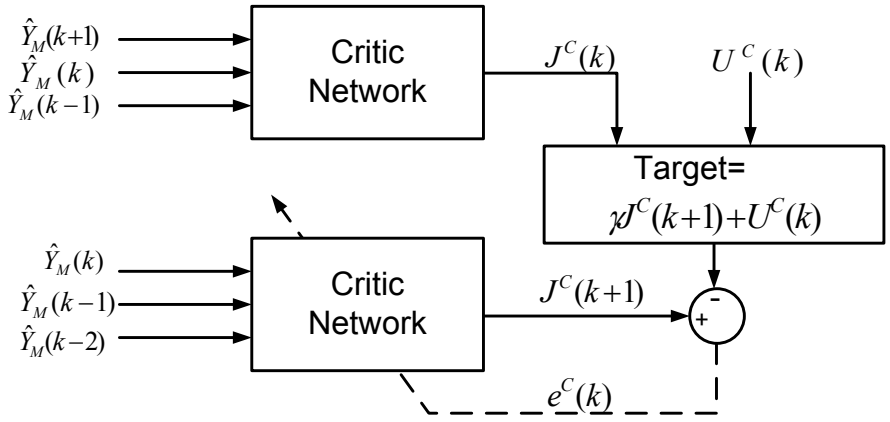


Figure 8.3: Training configuration of the critic network. $\hat{Y}_M(k)$ is the output of the model network at time *k*.

8.3.4 Design and Training of the Action Network

The objective of the action network is to find the optimal control vector *u**, to minimize J^{c}, thereby optimizing the overall cost expressed as a sum of all U^{c} over the time-horizon of the control problem. This is achieved by training the action network with an error vector $e_A(k)$, as defined in (8.12).

$$e_A(k) = \frac{\partial J^{\mathrm{c}}(k)}{\partial A(k)} \tag{8.12}$$

The derivative of the cost function $J^{\mathrm{c}}(k)$ with respect to the output of the action network $A(k)$ is obtained by backpropagating $\partial J^{\mathrm{c}}/\partial J^{\mathrm{c}}$ first through the critic network, and then through the model network to compute the desired adjustment of the output of the action network. This gives $\partial J^{\mathrm{c}}(k)/\partial \hat{Y}_M(k)$ and $\partial J^{\mathrm{c}}(k)/\partial \mathrm{A}(k)$ in Fig. 8.1 for the readout weights $W_A(k)$ and the output vector $A(k)$ of the action network. The readout update equations of the action network are shown as (8.13) and (8.14).

$$W_A(k+1) = W_A(k) + \Delta W_A(k) \tag{8.13}$$

To minimize $\frac{1}{2}e_A^2(k)$

$$\Delta W_A(k) = -\eta_A \cdot e_A(k) \cdot \frac{\partial e_A(k)}{\partial W_A(k)} \tag{8.14}$$

where η_A is a positive learning rate [118].

For traditional, non-spiking neural networks, for example, MLPs and RBFs, the calculation of partial derivatives in the error backpropagation process can be done using chain rules without any difficulties. However, when it comes to the BIANN, how to obtain $\partial J^{\mathrm{c}}(k)/\partial \hat{Y}_M(k)$ and $\partial J^{\mathrm{c}}(k)/\partial \mathrm{A}(k)$ becomes a non-trivial mathematical problem. Suppose the input to a BIANN is denoted as $I(k)$, and the output of the BIANN is denoted as $O(k)$, as shown in Fig. 8.4, then it is almost impossible to derive the closed form equations for $\partial I(k)/\partial O(k)$. The reason is that the Izhikevich model for spiking neurons and the mathematical expression of spike-timing-dependent-plasticity (STDP) in the BIANN contain non-continuous, non-differentiable terms. The detailed mathematical equations for the Izhikevich spiking neuron model and the STDP rule have been given in Chapter 6, in equation (6.1) and (6.3).

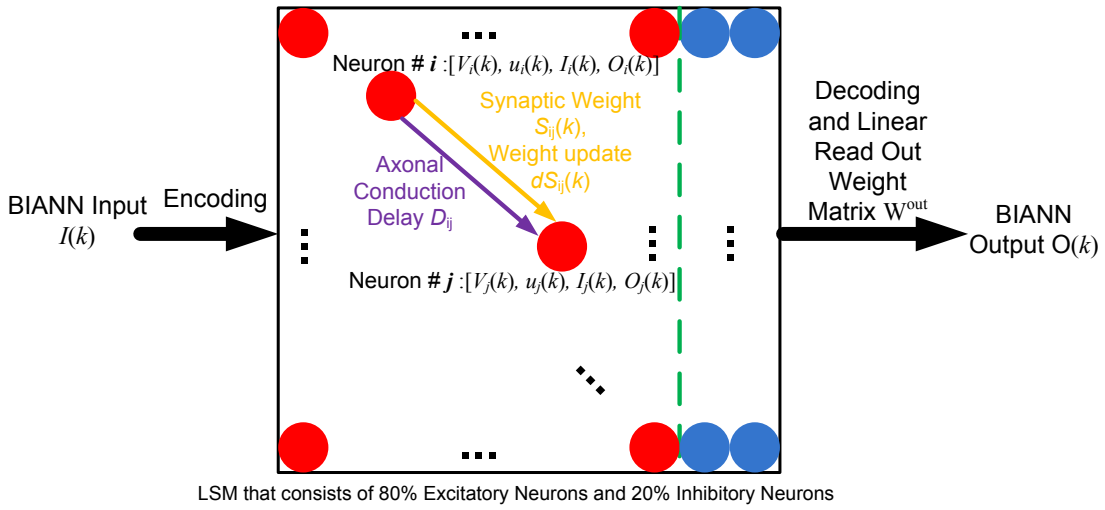


Figure 8.4: Variables used in a BIANN.

The differential equation (8.15), which is used to describe the behavior of each spiking neuron, contains a process of comparison and reset for variables V and u at each time step, which results in sudden jumps of the output even when the input to a neuron is continuous. As a result, analytical derivatives cannot be obtained for a BIANN.

$$\begin{cases} \frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I \\ \frac{du}{dt} = a(bV - u) \end{cases}$$

$$\text{if } V \geq 30 \text{ mV, then } \begin{cases} V = c \\ u = u + d \end{cases} \tag{8.15}$$

Therefore, a numerical derivative calculation algorithm for the BIANN is proposed to overcome this problem. The details of this algorithm are given later in Section 8.4.

8.3.5 Online Training Procedure of the HDP-BIANN Controller

The HDP-BIANN controller system consists of three BIANNs, namely, the model BIANN, the critic BIANN and the action BIANN. The online training procedure for the

HDP-BIANN controller consists of three training cycles: one for the model BIANN, one for the critic BIANN and one for the action BIANN.

One thing that should be emphasized here is that the HDP-BIANN controller runs online and the training never stops. This is very different from those well-known HDP-based control systems using the MLP or the RBF neural networks. In those control systems, a pre-trained, "universal" model network is first obtained. Then, during the iterative training process of the critic and action network, not only the model stays unchanged, most importantly, at the end of the training; all the weights in the critic network and action network converge to "optimal" values. Finally, the ready-to-use controller consists of three fixed-weights neural networks. In the case of HDP-BIANN controllers, the pre-trained model BIANN (using PRBS for pertubation, trained at several operating points) is only used as the initial network for the online training of the whole HDP-BIANN controller. In other words, the weights in the model BIANN will not be fixed. Moreover, the critc BIANN and the action BIANN are also updated continously. The flowchart of the whole training/running procedure of the HDP-BIANN controller is illustrated in Fig. 8.5.

The PSCAD code and MATLAB code for the online training and running of the HDP-BIANN controller are given with detailed explanations in the Appendix of this dissertation. The power system is simulated in PSCAD with interface to MATLAB, as shown in Appendix A. The HDP-BIANN controller is implemented in MATLAB and called by PSCAD through interface. The MATLAB code of the HDP-BIANN controller is given in Appendix B-D.

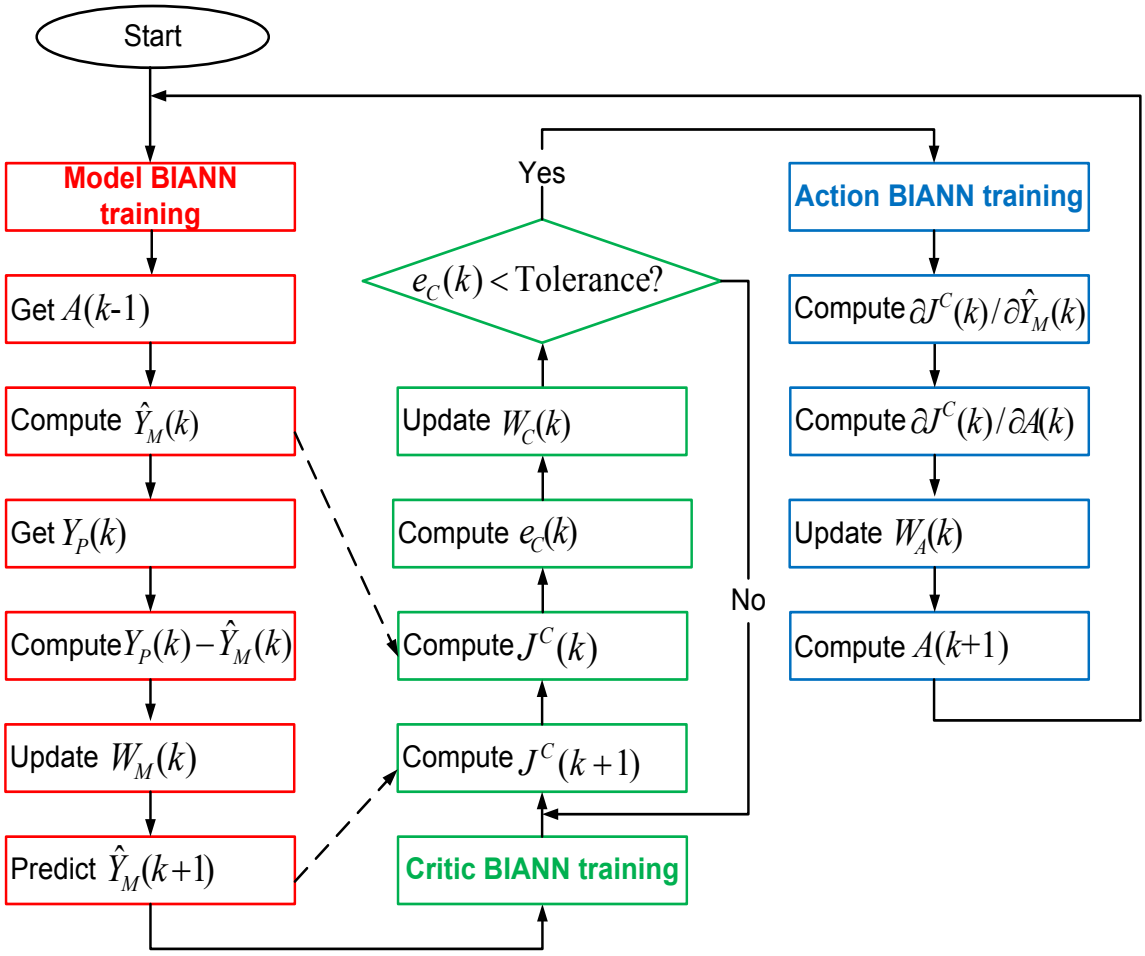


Figure 8.5: Training procedure of the HDP-BIANN controller.

8.4 Numerical Derivative Calculation Algorithm for Error-Backpropagation in BIANNs

It has been mentioned in Section 8.3.4 that the major technical challenge of using BIANNs in HDP is that there is no closed form mathematical equations for calculating the derivative of the output with respect to the input of the BIANN. There are two reasons why an analytical form of the derivative is difficult to derive: firstly, the Izhikevich spiking neuron model contains inequality, discontinuity and non-differentiable terms; secondly, the STDP rules in the BIANN causes changes in the weights, as a result,

the networks before and after error backpropagation are not exactly the same. Therefore, a numerical derivative calculation algorithm is proposed to solve the error backpropagation challenge for the BIANN. The dashed line in Fig. 8.6 shows the error backpropagation path through the critic BIANN, the model BIANN and finally the action BIANN. The ultimate goal is to accurately update the weights in the action BIANN, hence make the action BIANN give the optimal control command.

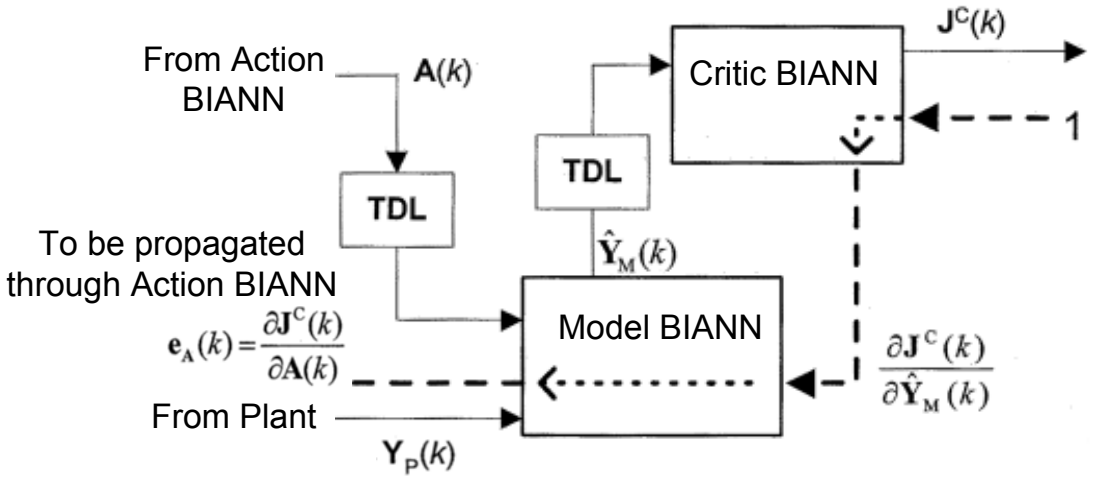


Figure 8.6: Error backpropagation path in the HDP-BIANN controller.

The update equation of the readout weights in the action BIANN has been given in (8.13) and (8.14), and by using chain rules, (8.14) can be rewritten as (8.16).

$$\Delta W_A(k) = -\eta_A \cdot \frac{\partial J^C(k)}{\partial A(k)} \cdot \frac{\partial e_A(k)}{\partial W_A(k)} = -\eta_A \cdot \frac{\partial J^C(k)}{\partial \hat{Y}_M(k)} \cdot \frac{\partial \hat{Y}_M(k)}{\partial A(k)} \cdot \frac{\partial e_A(k)}{\partial W_A(k)} \tag{8.16}$$

In (8.16), the term $\partial J^C(k)/\,\partial \hat{Y}_M(k)$ is the derivative of the output with respect to the input of the critic BIANN; and the term $\partial \hat{Y}_M(k)/\partial A(k)$ is the derivative of the output with respect to the input of the model BIANN. How to calculate these two derivatives numerically is illustrated in Fig. 8.7 (taking $\partial \hat{Y}_M(k)/\partial A(k)$ as an example).

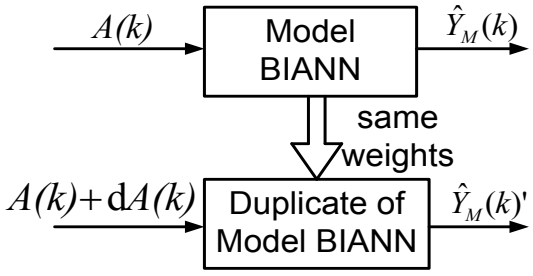


Figure 8.7: Numerical derivative calculation algorithm of the model BIANN.

At each sampling step, a duplicate of the model BIANN is created, which has the exact same set of weights as the model BIANN at this moment. A small vector, $dA(k)$, is added to the original input, $A(k)$, and then sent to the duplicate network. The small change in the input will result in a small change in the output, and the new output of the network is represented as $\hat{Y}_M(k)'$. According to the definition of derivate, as long as the change $dA(k)$ is sufficiently small, the numerical derivative of the network at this moment can be expressed in (8.17).

$$\frac{\partial \hat{Y}_M(k)}{\partial A(k)} = \frac{\hat{Y}_M(k)' - \hat{Y}_M(k)}{dA(k)} \tag{8.17}$$

Similarly, the numerical derivative $\partial J^c(k) / \partial \hat{Y}_M(k)$ can be calculated for the critic BIANN.

8.5 Case Studies and Control Performances

The MLPNN is a input-output system model, which generates the same output when given the same input, at any given time step. Different from MLPs, the BIANN is based on reservoir-computing technique. The internal "reservoir", in this case, the spiking neuron model, generates dynamic output in different patterns with respect to the input signals. Therefore, different from MLP-based technique, all the training of BIANN-

based HDP technique is achieved in an online fashion. The power system is first controlled by the conventional PID controller. The model network is trained to predict the behavior of the plant; the action network is trained to mimic the behavior of the PID controller; the critic network is trained to compute the cost-to-go function based on the prediction of the model network. This step is considered complete when the error of each neural network falls under a preset threshold.

After training the model, action and critic network online while the plant is controlled by PID controller, the HDP-based BIANN controller is ready to control the plant for the real-time operation. The performances of the optimal BIANN controller are compared with those of the conventional controller (CONVC) and those of the HDP-based controller using MLPs. The comparison considers system damping and transient stability. Two different types of disturbances, namely, a $\pm 5\%$ step change in the reference voltage of the exciter and a three phase short circuit at the infinite bus, are carried out to evaluate the performances of the controllers.

8.5.1 HDP-based Control using MLPs

For performances comparison purposes, an HDP-based controller using three MLPs is also developed for the same generator in the same SMIB power system of Fig. 8.2. The MLPs used for this application each consist of three layers of neurons interconnected by an input weight matrix W and an output weight matrix V. These three layers are the input, hidden and output layer, as shown in Fig. 8.8.

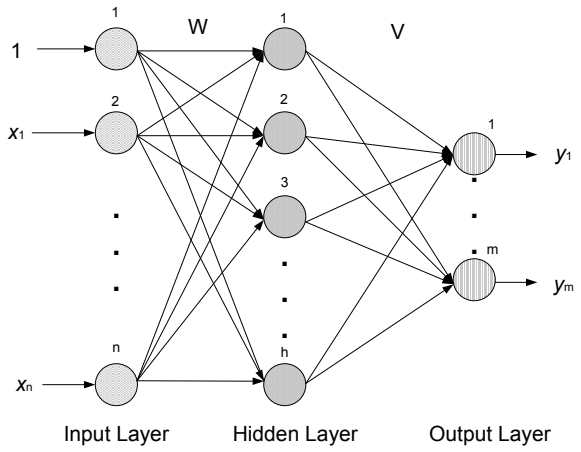


Figure 8.8: Structure of the MLP.

The weights of these MLPs are adjusted using the gradient-descent-based backpropagation algorithm [16]. The activation function for neurons in the hidden layer is given by the sigmoid function in (8.18).

$$f(x) = \frac{1}{1+e^{-x}} \tag{8.18}$$

The output layer neurons are formed by the inner products between the nonlinear regression vector from the hidden layer and the output weight matrix, V. The numbers of neurons in the hidden layer of the model MLP, the critic MLP and the action MLP are all 30. These values depend on a tradeoff between convergence speed and accuracy, and they are chosen by several trial runs. Since the number of neurons in the hidden layer of the MLPs are much smaller than the number of neurons in the dynamic reservoir of the BIANNs, the computational effort that are required by the HDP-MLP controller are much less than that are required by the HDP-BIANN controller.

The same utility function is used for this HDP-MLP controller, as defined in (8.9).

The training process of the model MLP, the critic MLP and the action MLP in this study follows the steps described in [120].

8.5.2 Case 1: ±5% Step Changes in the Reference Voltage of the Exciter

First, the plant is operating at a steady-state condition, controlled by CONVC. At t=5 s, the control is switched from CONVC to the HDP-based MLP controller or the HDP-based BIANN controller. In both cases the two types of HDP-based neurocontrollers drive the plant back to the steady state. Then, at t=20 s, a 5% step increase in the reference voltage of the exciter is applied, and at t=30 s, the 5% step increase is removed, and the system returns to its initial operating point. The results in Fig. 8.9 and Fig. 8.10 show that the HDP-BIANN controller increases the transient system damping compared to the CONVC and the HDP-MLP controller. The simulation time step used in PSCAD is 10 microseconds and the convergence threshold of the training of three BIANNs in the HDP-BIANN controller is 0.001%.

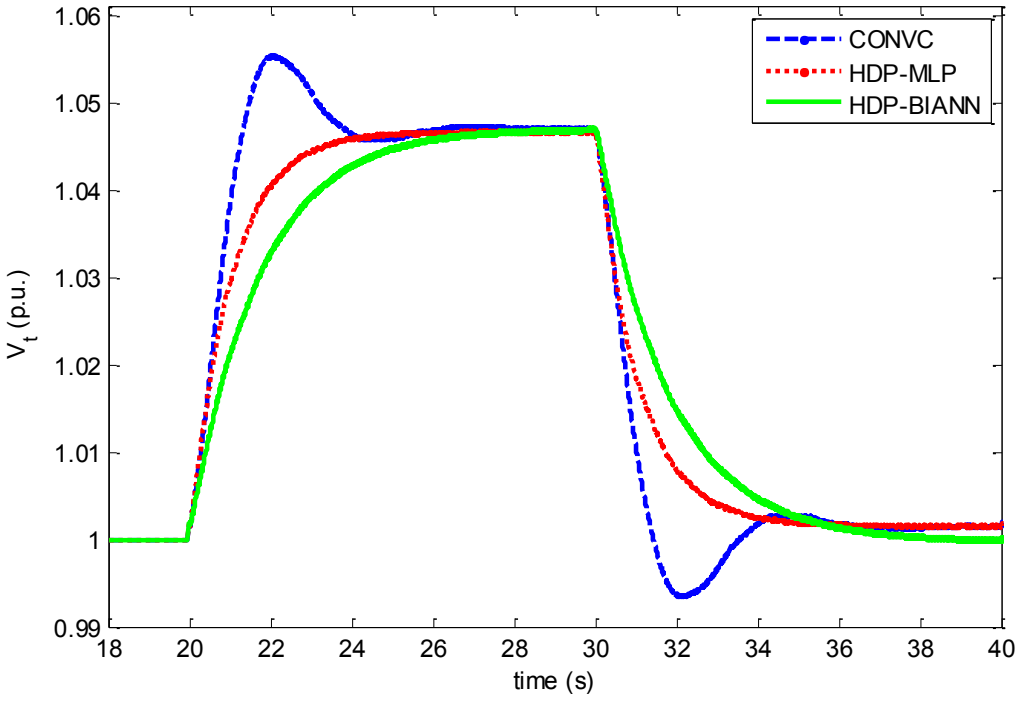


Figure 8.9: Terminal voltage as a function of time for ±5% step changes in reference voltage of the exciter.

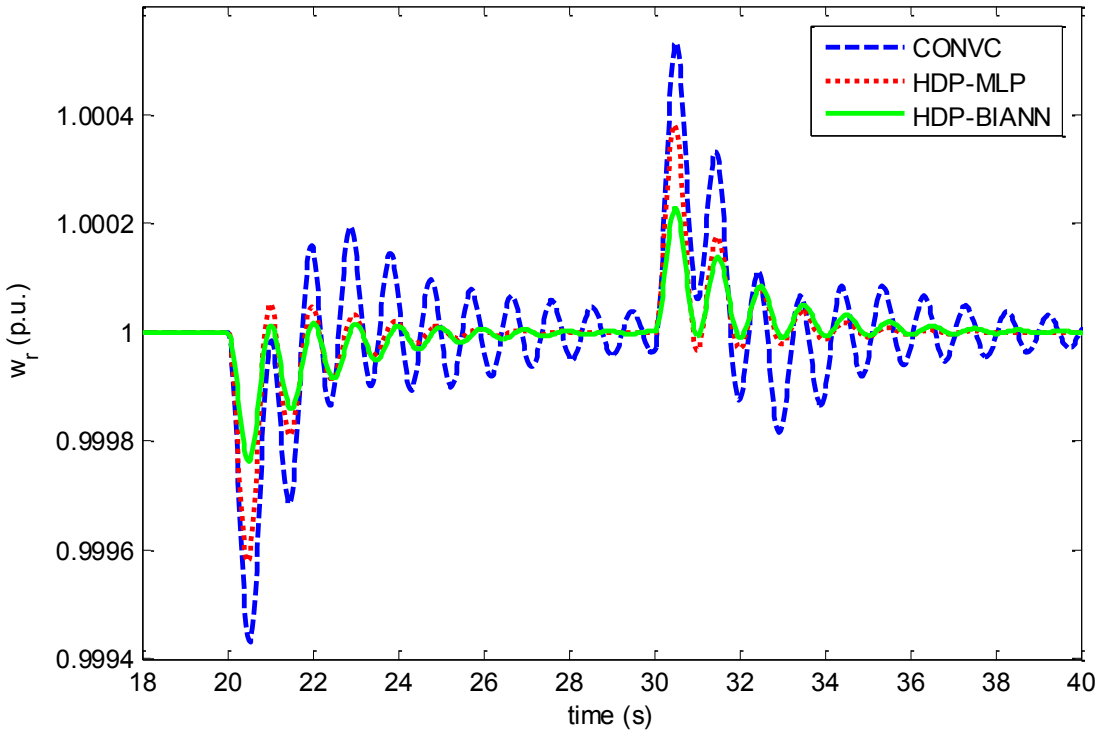


Figure 8.10: Rotor speed as a function of time for $\pm 5\%$ step changes in reference voltage of the exciter.

The difference between the responses for terminal voltage (Fig. 8.9) and rotor speed (Fig. 8.10) has to do with the fact that terminal voltage can change much more rapidly than rotor speed. The terminal voltage is determined partly by the field current and automatic voltage regulator (AVR), but to a large part by the network to which the terminal voltage point is connected. Rotor speed is much more dependent on the parameters and the states and the controls of the generator and exactly when the field current is increased and decreased by the AVR. In addition, the choice of the utility function being optimized will affect the difference between terminal voltage and rotor speed responses.

In the terminal voltage (Fig. 8.9), the HDP-BIANN curve responds slower, thus it is more damped than the HDP-MLP and the CONVC. In the rotor speed (Fig. 8.10), the

HDP-BIANN decays faster than the other two; therefore the system again has more damping for the HDP-BIANN. The HDP-BIANN thus increases the damping shown in both sets of curves. The overshoots in the rotor speed are 0.00022 p.u., 0.00042 p.u. and 0.00056 p.u. for the HDP-BIANN, HDP-MLP and CONVC, respectively. The HDP-BIANN thus outperforms the HDP-MLP and the CONVC in both damping and minimizing the impact of system disturbances.

The result of the critic network's online training is shown in Fig. 8.11. During the training, the $\pm 5\%$ step change is applied to the system repeatedly every 10 seconds starting from t=20 s, (at t=20 s, 30 s …). The output of the critic network, J, converges fast after only a few seconds. When the 5% step change happens every 10 seconds, J at first increase rapidly but then decreases quickly because of the training of the critic network. These results also show that the J of the HDP-BIANN decreases much quicker than the J of the HDP-MLP, and this is because the HDP-BIANN's training converges much faster than that of the HDP-MLP.

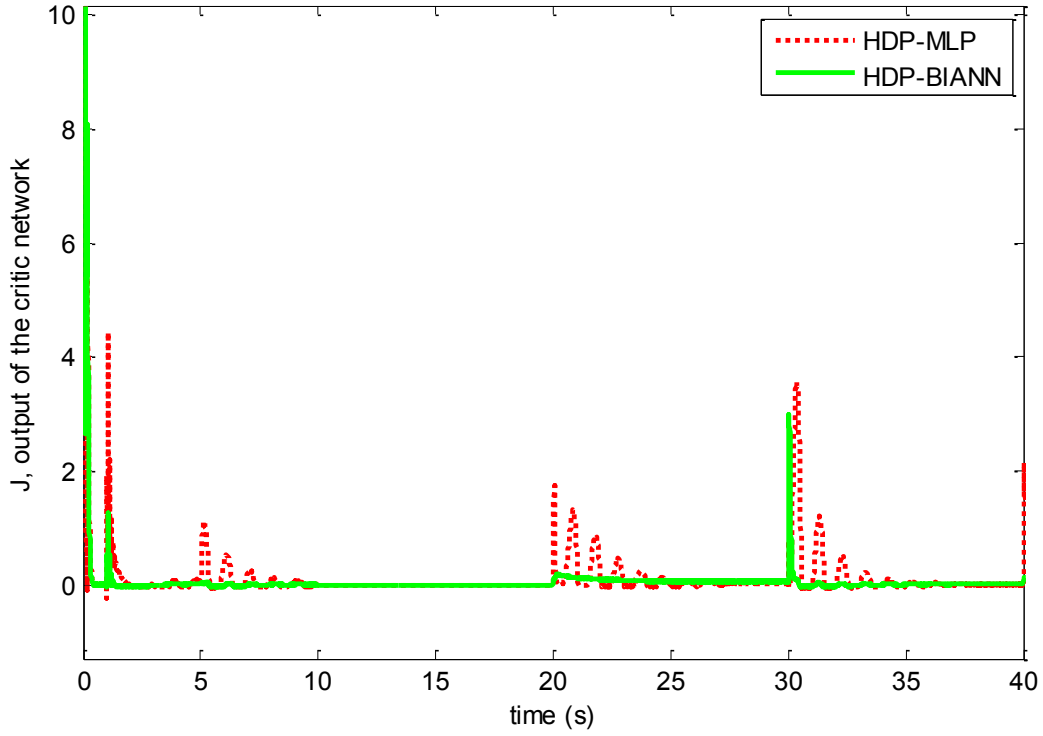


Figure 8.11: Output of the critic network versus training time in case 1.

8.5.3 Case 2: Three-Phase Short Circuit Test at the Infinite Bus

A large disturbance is now given to the system to evaluate the performances of the HDP-based BIANN controller. First, the plant is operating at a steady-state condition, controlled by CONVC. At t=5 s, the control is switched from CONVC to the HDP-based MLP controller or the HDP-based BIANN controller. Then, at t=10 s, a temporary three-phase short circuit is applied at the infinite bus for 100 ms from t=10 s to 10.1 s. The results in Fig. 8.12 and Fig. 8.13 show that the HDP-BIANN controller improves the transient system damping compared to the CONVC and the HDP-MLP controller. The HDP-based BIANN controller damps out the oscillations for the terminal voltage (V_t) and the rotor speed (ω_r) more effectively than the CONVC and the HDP-based MLP controller. The overshoots in the terminal voltage are 0.032 p.u., 0.052 p.u. and 0.056 p.u. for the HDP-BIANN, HDP-MLP and CONVC, respectively. It is clearly shown that the

HDP-BIANN outperforms the HDP-MLP and the CONVC in terms of controlling both the terminal voltage and the rotor speed to have smaller overshoots and settling back to pre-fault values quicker..

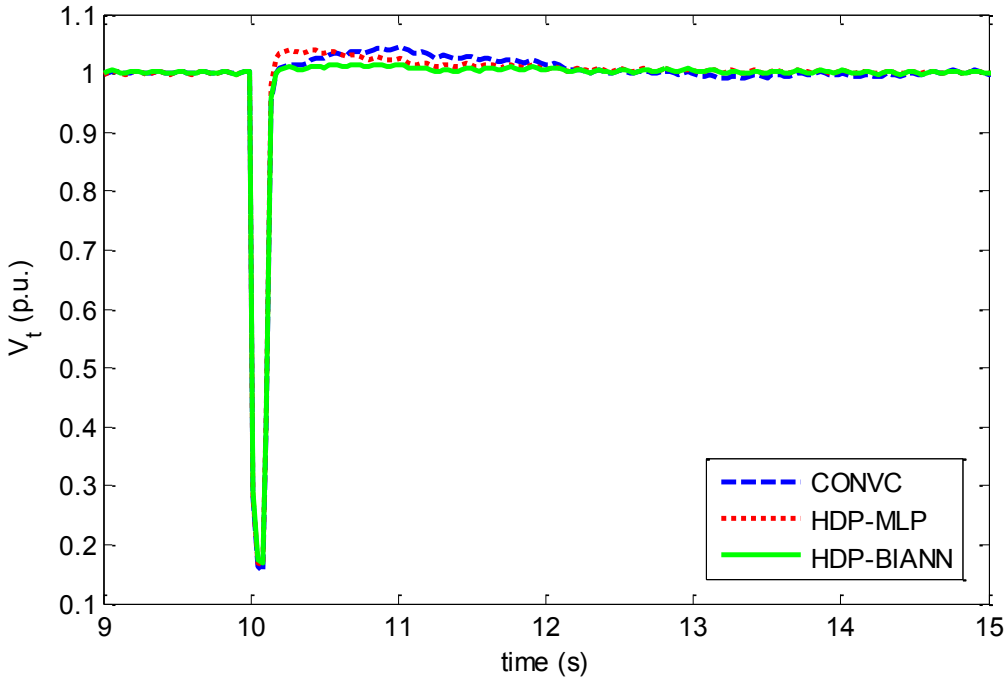


Figure 8.12: Terminal voltage as a function of time following a three-phase short circuit test.

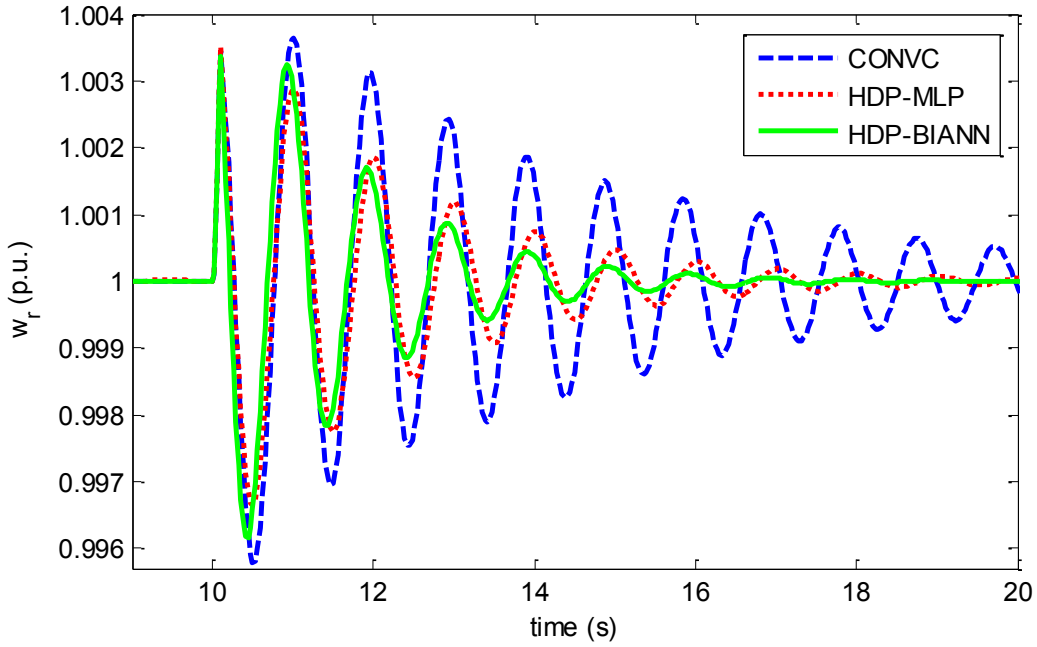


Figure 8.13: Rotor speed as a function of time following a three-phase short circuit test

The result of the critic network's online training is shown in Fig. 8.14.

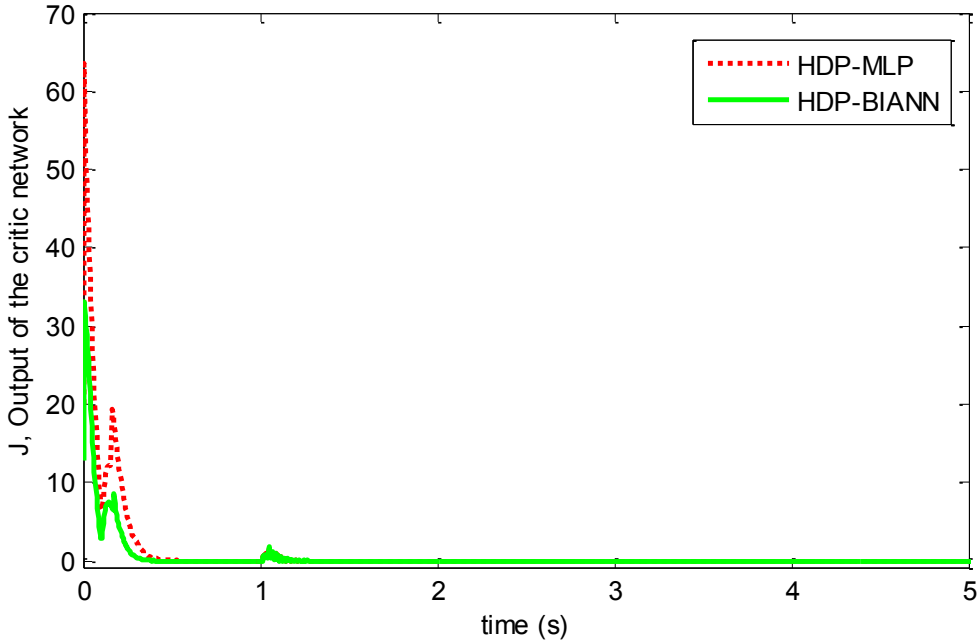


Figure 8.14: Output of the critic network versus training time in case 2.

8.5.4 Comparison between HDP-MLP and HDP-BIANN controllers

Both HDP-MLP controller and HDP-BIANN controller are based on the same principle. However, it is clearly observed in all the case studies that HDP-BIANN controller outperforms the HDP-MLP controller in terms of response time and control accuracy. This is due to the fact that MLP is an "input-output" neural network, which is capable of modeling the system as an "input-output" system. The MLP gives the same output when fed with the same input at any given time. On the contrary, the BIANN is a dynamic neural network, which mimics the behavior of the system with consideration of various system dynamics, not only dependent on the instantaneous input of the system. The power system is a typical dynamic system. As a result, the BIANN fits better than the MLP for heuristic dynamic programming, providing more accurate modeling and control of the power system.

Although outperforming MLP-based controller in all the case studies, the computational effort required by the BIANN-based controller is significantly higher than the MLP-based controller. This is due to the fact that large amount of computation is required in the spiking neuron model to generate the dynamic responses of the BIANN. The BIANN is based on the spiking neural model, which mimics the behavior of living neurons. The same time step is used in simulating the BIANN and the power system, which means the same control technique can be potentially extended to living neurons in real time.

8.6 Chapter Summary

This chapter has shown that a BIANN can be used in adaptive critic neural network design for optimal control problems in a small power system. An HDP-based optimal

controller using three BIANNs has been designed for the control of a turbo generator in a single machine infinite bus power system.

The difficulties in backpropagating errors through a BIANN have been overcome by developing a numerical derivative calculation algorithm. The HDP-BIANN controller for the turbo generator in the SMIB system is trained and run online for different operation conditions. The results show that the HDP-BIANN controller improves the system damping as well as dynamic transient stability more effectively than the conventional controllers (CONVC) for not only small step changes, but also large disturbances such as a three phase short circuit.

This is the first time that BIANN is used for closed-loop optimal control application in a power system.

CHAPTER 9 CONCLUSIONS, CONTRIBUTIONS AND RECOMMENDATION FOR FUTURE WORKS

9.1 Summary

The objective of this research is to develop a novel type of brain-like artificial neural network (ANN), i.e. a biologically inspired artificial neural network (BIANN), for modeling and control of a power system.

Chapter 1 introduced the background information related to the proposed research. Worldwide concern about the environmental pollution and a possible energy crisis has led to increasing interests in innovative technologies for generation of clean and renewable electrical energy. All the emerging technologies have brought new challenges to the power quality and power system operation, which calls for more advanced monitoring and control technologies for the power system. Computational intelligence techniques, such as artificial neural networks (ANNs), have been widely used to improve the performance of power system monitoring and control. Although inspired by the neurons in the brain, ANNs are largely different from living neuron networks (LNNs) in many aspects. Due to the oversimplification of the ANN, the huge computational potential of LNNs cannot be realized by ANNs. Therefore, a more brain-like artificial neural network is desired to bridge the gap between ANNs and LNNs. The main scope of this research is to develop a biologically inspired artificial neural network (BIANN), which is not only biologically meaningful, but also computationally powerful. The BIANN will serve as a novel computational intelligence tool in monitoring, modeling and control of the power systems.

Chapter 2 presented a comprehensive survey of the previous work on the application of conventional ANNs for power system operation, monitoring and control. Applications of the Echo state network (ESN), which is a form of reservoir computing since the year 2002, are reviewed in detail.

Chapter 3 summarized the previous work related to the BIANNs, which includes most commonly used neuronal coding methods and spiking neuron models in the BIANN research area. A comparison of these commonly used neuron models has been presented to better understand the advantages and challenges of each neuron model to serve as an intelligence tool in monitoring, modeling and control applications.

To further explore the capability of reservoir-computing-based artificial neural networks, Chapter 4 proposed an ESN-based approach to predict the true harmonic contributions of nonlinear loads in the power grid. The performances of three different types of ANNs, for the same application have been compared, namely MLPs, RNNs and ESNs. The ESNs clearly outperform the other two types of ANN in terms of modeling accuracy and computational requirements.

As a follow-on to Chapter 4, Chapter 5 introduced an ESN-based indirect adaptive control scheme for an active filter and validated this by simulation in order to eliminate the harmonic currents injected by those nonlinear loads. It is clearly shown that the ESN-based control scheme outperforms the conventional linear controller PI controller in terms of control accuracy and response time.

With the performances of the ESNs in system modeling and control applications demonstrated, Chapter 6 considered a novel BIANN architecture based on the idea of reservoir computing. A mean-firing-rate-based coding method has been explained in

Chapter 6. With this coding method and the carefully chosen Izhikevich Model as the spiking neuron model, the proposed BIANNs can be trained online for modeling purposes. Simulation results of using the BIANN as a function approximator have also been given in Chapter 6.

To further investigate the capability of BIANNs for modeling in power system applications, a BIANN-based modeling approach for modeling performance and behavior of a generator in a single-machine infinite-based system (SMIB) has been presented in Chapter 7. The feasibility of the BIANN-based modeling scheme has been shown through simulation under various conditions. It has been clearly shown that the proposed BIANN-based modeling scheme for generators can accurately identify the dynamics of the SMIB power system and can be potentially used for further control applications.

With the modeling capability of BIANNs demonstrated, a HDP (heuristic dynamic programming)-based optimal controller using BIANNs has been developed for controlling of generators in a SMIB system in Chapter 8. The performances of the HDP-BIANN control schemes have been evaluated in various operating conditions and disturbances in the SMIB system and provide excellent damping results. A comparison of control performances between the HDP-BIANN controller and the HDP-MLP controller is also carried out in Chapter 8, showing that the HDP-BIANN control schemes outperform the conventional linear controller in all tested conditions.

In summary, a novel reservoir-computing-based artificial neural network, i.e. biologically-inspired artificial neural network (BIANN), has been proposed to bridge the gap between conventional artificial neural network and living neural networks. BIANN-based modeling and control approaches have been proposed for power system

applications and validated through simulation. It is clearly shown that the proposed BIANN-based control scheme can provide more advanced control for the power system and thus enhance the reliability of the power system.

9.2 Contributions

The literature review and research work presented in this thesis have resulted in several publications, listed as follows:

[1] **J. Dai**, G. K. Venayagamoorthy, R. G. Harley and S. M. Potter, "Reservoir-computing-based, Biologically-inspired Artificial Neural Network for Modeling of a Single Machine Infinite Bus Power System," *IEEE World Congress on Computational Intelligence (WCCI'12)*, Brisbane, Australia, June10-15, 2012.

[2] **J. Dai**, G. K. Venayagamoorthy, Ronald G. Harley and Keith A. Corzine, "Indirect Adaptive Control of an Active Filter Using Echo state networks", *Proc. of the IEEE Energy Conversion Congress&Expo(ECCE'10)*, Atlanta, USA, Sep12-16, 2010, pp.4068 - 4074.

[3] **J. Dai**, G. K. Venayagamoorthy and R. G. Harley, "An Introduction to the Echo state network and its Applications in Power System," *Proc. 15th International Conference on Intelligent System Applications to Power Systems* (*ISAP'09*), Curitiba, Brazil, Nov8-12, 2009, pp. 1-7.

[4] **J. Dai**, G. K. Venayagamoorthy and R. G. Harley, "Harmonic Identification using an Echo state network for Adaptive Control of an Active Filter in an Electric Ship," *Proc. of the IEEE Int. Joint Conf. on Neural Networks(IJCNN'09)*, Atlanta, June23-26, 2009, pp. 634-640.

[5] **J. Dai**, P. Zhang, J. Mazumdar, R. G. Harley and G.K. Venayagamoorthy, "A Comparison of MLP, RNN and ESN in Determining Harmonic Contributions from Nonlinear Loads," *Proc. of the IEEE 34th Annual Conf. on Industrial Electronics*(*IECON'08*), Orlando, Nov10-13, 2008, pp.3025 –3032.

[6] J. Liang, **J. Dai**, G. K. Venayagamoorthy and Ronald G. Harley, "Dynamic System Eigenvalue Extraction using a Linear Echo state network for Small-signal Stability Analysis - a Novel Application", *Proc. of the IEEE Int. Joint Conf. on Neural Networks (IJCNN'10)*, Barcelona, Spain, July18-23, 2010, pp. 1 - 8.

[7] P. Zhang, Y. Du, **J. Dai**, T. G. Habetler, and B. Lu, "Impaired Cooling Condition Detection Using DC Signal Injection For Soft-Starter-Connected Induction Motors," *IEEE Trans. on Industrial Electronics*, vol.56, no.11, pp.4642-4650, Nov. 2009.

In addition the following paper is in preparation:

[8] **J. Dai**, G. K. Venayagamoorthy, R. G. Harley and S. M. Potter, "Adaptive Critic-Design-Based Optimal Control for a Turbo Generator in a Single Machine Infinite Bus Power System using Biologically Inspired Artificial Neural Networks," *IEEE Transactions on Neural Networks*.

The main contributions of this research are summarized as following:

1. A comprehensive survey of the application of artificial neural networks in power systems is presented. Various artificial neural networks techniques are reviewed in terms of their application as well as their effectiveness for applications in power systems.

2. A comprehensive survey of the coding approaches and the spiking neuron models of living neurons is also presented. These coding approaches and models are compared in terms of effectiveness, biological resemblances and computational complexity.

3. An ESN-based true load harmonic current identification approach is proposed for predicting the true current harmonics attributed to the nonlinear load. This approach can help evaluate the source of harmonic pollution in power system. It is shown that the ESN outperforms conventional ANNs in both performance and computational effectiveness.

4. An ESN-based indirect adaptive control scheme is proposed for advanced control of shunt active filters. This control scheme can effectively remove the harmonics in power system and better improve the power quality.

5. A novel type of artificial neural network, the BIANN is proposed to bridge the gap between ANNs and LNNs. The BIANN has a more brain-like architecture compared to ANNs, which can better emulate the behavior of LNNs and assist the study of LNNs in terms of training approaches. Eventually, it may be possible to replace the network of artificial spiking neurons in a BIANN with LNNs using similar schemes to employ the huge computational potential of the Living Neural Networks.

6. The BIANN framework which includes feasible encoding, decoding and training algorithms has been developed. Under this framework, the BIANN can be successfully used to model nonlinear systems.

7. A novel numerical derivative calculation algorithm has been proposed to address the error backpropagation problem in BIANNs, which has made it practical to use BIANNs in closed-loop control applications.

8. An HDP-based optimal control scheme using BIANNs has been developed for a turbo generator in single machine infinite bus power system to replace conventional linear controllers. The performances of the HDP-BIANN controller under small and large

disturbances again prove that the BIANN is a powerful new type of ANN and can be used in closed-loop control applications.

9. The HDP-BIANN control has been compared with the HDP-MLP control for the generator in a single machine infinite bus power system, and it has been shown that the HDP-BIANN controller increases the transient system damping compared to the HDP-MLP controller. .

9.3 Recommendations for Future Work

The primary target of this research is to develop a type of more brain-like artificial neural network and bridge the gap between living neural networks and artificial neural networks. The newest findings and developments in the field of neuroscience helps researchers understand how the neurons and the brain work in a real world, hence more accurate spiking models of a single neuron and a group of neurons are created. These biologically-plausible spiking neuron models can be used to create more brain-like BIANN architectures.

The work of this thesis addressed several key issues of using BIANNs for nonlinear system modeling and control applications; however, there are several directions in which further research could build on the results presented in this work, including: Real-time implementation of the BIANN; investigation to the scalability of the BIANN for larger system modeling and control; replacing the BIANN with real living neural networks and performing in-vitro experiments.

9.3.1 Real-time Implementation of the BIANN

An HDP-BIANN optimal controller for a turbo generator in a single machine infinite bus power system is developed in Chapter 8. The generator and the power system models

are built using the PSCAD simulation platform, and the adaptive-critic-design-based control algorithm using three BIANNs are written in FORTRAN and MATLAB code. The simulation and control are not done in real time.

Whether the BIANN can be used in a real-time environment still needs to be investigated. Take the control application in Chapter 8 as an example, the simulation of the power system can be done in a real time digital simulator (RTDS) using RSCAD software, and the control algorithm can be developed on a DSP chip interfaced with the RTDS. This research direction is essential to evaluate the computational efficiency of the BIANN.

As mentioned before, the ultimate goal of developing the BIANN is to create a more powerful computational tool. If the BIANN is actually more biologically meaningful and functions just like how the human brain works, it should be able to deal with real-time decision-making tasks.

Implementing BIANNs in a real-time environment may face the following challenges:

Compared to activation-function-based, non-spiking artificial neural networks, additional encoding and decoding processes are required for the BIANN during the calculations, which increase the computational burden of using BIANN;

The size of the BIANN is usually much larger than the commonly used MLPs and RNNs. The number of spiking neurons in the dynamic reservoir of the BIANN can easily go to 1000, which will inevitably result in longer computational time;

Unlike the ESN, the weights in the dynamic reservoir in the BIANN are not fixed. Instead, as long as there are inputs being fed into the BIANN, the weights in the dynamic

reservoir change continuously following the rule of spike-timing-dependent-plasticity. These changes represent the evolving and learning of the BIANN, but at the same time increase the computational complexity.

9.3.2 Scalability: Using BIANNs in More Complicated System

So far, the BIANN has only been used to model and control a relatively small system, i.e. the single machine infinite bus system. The dimensionality of the control problem is not large (only 2-input-2-output).

Whether the BIANN can be used for more complicated modeling and control problems is another valuable research direction. One of the driving forces of searching for more advanced neural network architecture is that MLPs might fail to converge when the system being modeled or controlled is too large.

Since the main framework of the BIANN has already been developed in this thesis, it can be transplanted to other power system applications such as wide area monitoring of a large-scale power grid.

9.3.3 From BIANN to MEA: Using Living Neural Networks

A system of multi-electrode array (MEA) culture plus embodiment has been developed in [22, 73]. As shown in Fig. 9.1 [122], the MEA culture is placed in an incubator to maintain the health of the living network. A living neuronal network (LNN) is cultured on the MEA where its activity is recorded, processed in real time, and used to control a robotic or simulated embodiment. The robot's input from proximity sensors is converted into electrical stimuli that are fed back to the neuronal network within milliseconds via a custom multi-electrode stimulation system. This closed-loop can achieve a certain level of control, for example, the drawing direction of the robot arm.

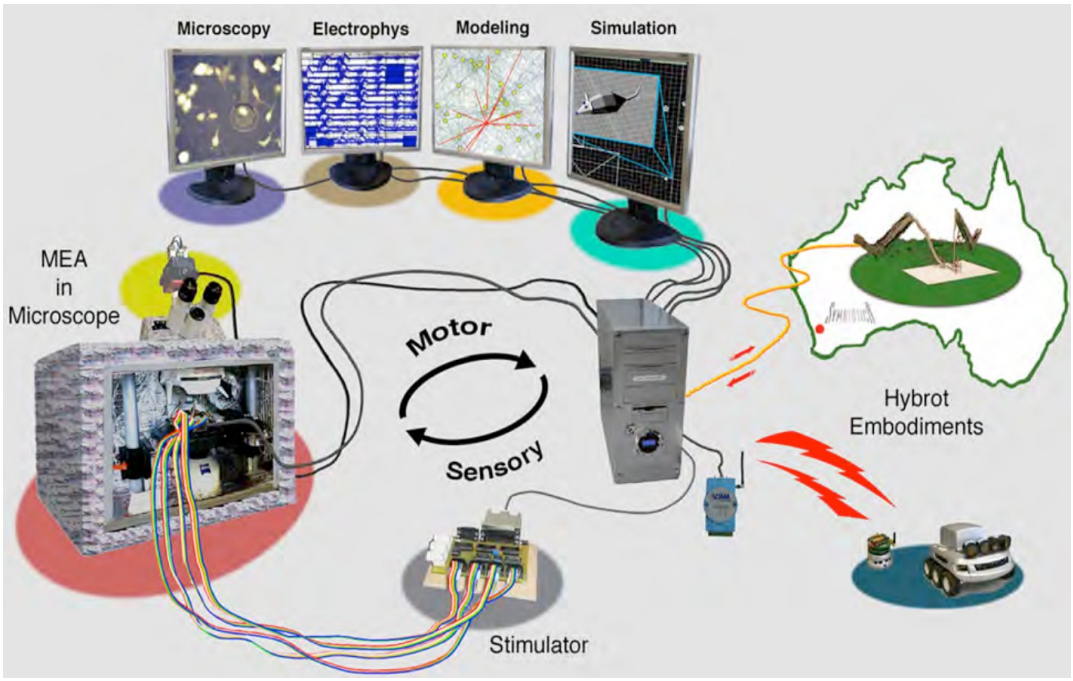
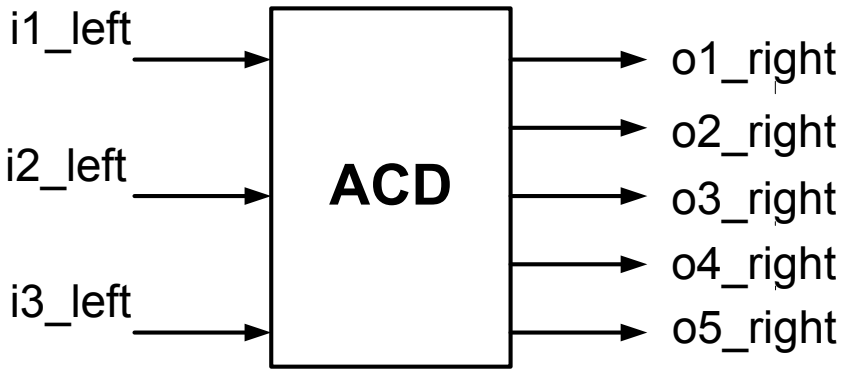


Figure 9.1: Hybrots: closed-loop embodied cultured networks [118].

Now, since the BIANN framework has been developed and precise training algorithm of the BIANN is available, the training algorithm of the BIANN can provide guidance to the stimulation of the living neural network culture growing on the MEA. It is promising that the closed-loop MEA system can be trained for more complicated tasks.

APPENDIX A: FORTRAN-MATLAB INTERFACE CODE IN PSCAD

This piece of FORTRAN code is used in the self-defined module named "ACD" to implement the HDP-BIANN controller in Chapter 8. This code builds an interface between PSCAD and MATLAB. The "ACD" module has three inputs and five outputs, as shown in the figure below.



The first two inputs of the module are the terminal voltage deviation (ΔV_t) and speed deviation ($\Delta\omega_r$) of the generator in the single machine infinite bus system, and the third input is the system time.

The first two outputs are the reference values of the turbine input active power (P_{in}) and exciter input voltage (V_{ref}); the third and fourth outputs are the outputs of the model BIANN and the fifth output of the module is the output of the critic network, *J*.

The online training of the HDP-BIANN controller is realized by a MATLAB file called "ACD_main.m", which will be given in Appendix B.

```
#STORAGE REAL:8

      STORF(NSTORF)   = $i1_left
      STORF(NSTORF+1) = $i2_left
      STORF(NSTORF+2) = $i3_left

CALL MLAB_INT("C:\Users\Matlab\", "ACD_main", "R R R",
"R(5)")

      $o1_right = STORF(NSTORF+3)
      $o2_right = STORF(NSTORF+4)
      $o3_right = STORF(NSTORF+5)
      $o4_right = STORF(NSTORF+6)
      $o5_right = STORF(NSTORF+7)


       NSTORF = NSTORF + 8
```

APPENDIX B: MAIN MATLAB CODE FOR HDP-BIANN CONTROLLER

This piece of MATLAB code named "ACD_main.m" implements the online training and running of the HDP-BIANN controller.

Two functions are used and called in this main file.

The first function is called "BIANN.m". It realizes the encoding, Izhikevich model for spiking neurons, dynamic reservoir weights update using STDP, decoding and readout weights update for the BIANN. The code is given in Appendix C.

The second function is called "derivative_BIANN.m". It realizes the numerical derivative calculation algorithm for the BIANN. The code is given in Appendix D.

```
function [u] = ACD_main( yout1,yout2, time)
%%used functions are: BIANN (input, error, network number (1 for model,
2
%%for critic, 3 for control), update (1 for yes, 0 for no), time).  the
function
%%first update weights then predict new output;
% derivative_BIANN(input, network number (1 for model, 2 for critic, 3
for control)) outputs are numerically computed derivative matrix

%load data from previous step
load('datastorage.mat');
% scale system output
output(1)=(yout1+0.01).*10;
output(2)=(yout2+0.0006).*1000;

%% model network
%desired output of model network
y_model=output;
%model network error
error_model=y_model-yhat(4,:);
%update BIANN with error and predict next step
yhat(4,:)=BIANN(input_current, error_model, 1, 1, time);

%% critic network
%input to critic network
input_c1=[yhat(2:4,1);yhat(2:4,2)];
input_c2=[yhat(1:3,1);yhat(1:3,2)];
```

```
%utility function
U=4*yout1^2+4*deltaV(1)^2+1.6*deltaV(2)^2+0.4*yout2^2+0.4*deltaw(1)^2+0
.16*deltaw(2)^2;
%check critic network output and compute error; no weights update at
this stage
J1=BIANN(input_c1, 0, 2, 0, time);
J2=BIANN(input_c2, 0, 2, 0, time);
error_critic=0.7*J1+U-J2;
%update network if does not converge
while error_critic>1e-2
    %update weights and compute J1 again
    %only output weights are updated and synaptic weights not updated
    J1=BIANN(input_c1, error_critic, 2, 1, time);
    %compute J2 again with the same weights
    J2=BIANN(input_c2, 0, 2, 0, time);
    %recompute error; repeat if not converging
    error_critic=0.7*J1+U-J2;
end

%% control network
%numerical derivative of critic network
derivative_critic=derivative_BIANN(iput_c2, 2, time);
%numerical derivative of model network
derivative_model=derivative_BIANN(input_current, 1, time);
%compute overal error for control network
error_control=derivative_critic(1,:).*derivative_model;
%update and predict next step output of control signal
input_current=BIANN(y_model,error_control, 3, 1, time);

%%scale control output and model prediction back to normal scale
u1=input_current(1)+1.3;
u2=input_current(2)/100+0.71;
y1=yhat(4,1)/10-0.01;
y2=yhat(4,2)/1000-0.0006;

%%shift variable for next step calculation
input_pre=input_current;
deltaV(2)=deltaV(1);
deltaV(1)=yout1;
deltaw(2)=deltaw(1);
deltaw(1)=yout2;
yhat=circshift(yhat,-1);

%%output matrix to PSCAD
u=[u1 u2 y1 y2 J2]

%%save data for next step
save datastorage input_current input_pre deltaV deltaw yhat J2

end
```

APPENDIX C: MATLAB CODE FOR FORWARD CALCULATION IN BIANN

File name: BIANN.m

```
function [ output ] = BIANN( input, error, network_number, update, time)
% network number =1 for model; =2 for critic, =3 for action
% update=1 if update required

%load each BIANN information
switch network_number
    case 1
        load('BIANN_model.mat');
    case 2
        load('BIANN_critic.mat');
    case 3
        load('BIANN_control.mat');
end

m=length(input);
n=length(error);

%%encoding
%compute required "1"s
input_encoded=circshift(input_encoded, [-1,0]);
y_residual=input(m)-1/200*sum(sum(input_encoded(1:19,:)));
number_of_input_needed=round(200*y_residual);
%cannot be lower than 0 or higher than 10
number_of_input_needed=max(min(number_of_input_needed,10),0);
%randomly select input channel as "1"
if number_of_input_needed>1
    index_neuron=randsample(10,number_of_input_needed);
    input_encoded(20,index_neuron)=1;
end


%%spiking network model
%run only if simulation moves in time step
if t~=mod(time*1000,1000)+1;
t=mod(time*1000,1000)+1;


%initial input
    Iin=20.*input_encoded(20,:);
    I=zeros(N,1);                    % initialize thalamic input
    for ii=1:input_number
        I(ii)=I(ii)+Iin(t);
    end
%find fire neurons and reset
```

```
    fired = find(v>=30);                % indices of fired neurons
    v(fired)=-65;
    u(fired)=u(fired)+d(fired);
%compute STDP and sd
    STDP(fired,t+D)=0.1;
    for k=1:length(fired)
      sd(pre{fired(k)})=sd(pre{fired(k)})+STDP(N*t+aux{fired(k)});
    end;
    firings=[firings;t*ones(length(fired),1),fired];
    k=size(firings,1);
    while firings(k,1)>t-D
      del=delays{firings(k,2),t-firings(k,1)+1};
      ind = post(firings(k,2),del);
      I(ind)=I(ind)+s(firings(k,2), del)';
      sd(firings(k,2),del)=sd(firings(k,2),del)-1.2*STDP(ind,t+D)';
      k=k-1;
    end;
    v=v+0.5*((0.04*v+5).*v+140-u+I);    % for numerical
    v=v+0.5*((0.04*v+5).*v+140-u+I);    % stability time
    u=u+a.*(0.2*v-u);                   % step is 0.5 ms
    STDP(:,t+D+1)=0.95*STDP(:,t+D);     % tau = 20 ms
  end;

%update synaptic weights every 1 sec
if t=1000
  STDP(:,1:D+1)=STDP(:,1001:1001+D);
  ind = find(firings(:,1) > 1001-D);
  firings=[-D 0;firings(ind,1)-1000,firings(ind,2)];
  s(1:Ne,:)=max(0,min(sm,0.01+s(1:Ne,:)+sd(1:Ne,:)));
  sd=0.9*sd;
end
%output matrix for BIANN
firingdata=circshift(firingdata, -1,0);
for i=1:length(fired)
    firingdata(20,fired+input_number)=1;
end
firingdata(20,1:input_number)=input_encoded(20,:);


%decode for training
analog_output=sum(firingdata)./20;
end

%% training
%update weights and output
    delta_weights=lg*error.'*analog_output.';
    weights=weights+delta_weights;
    output=weights*analog_output;


%save all infomraiton if update=1, otherwise do not save any
information
    if update==1
        switch network_number
            case 1
                save BIANN_model;
```

```
            case 2
                save BIANN_critic;
            case 3
                save BIANN_control;
        end
    end

end
```

APPENDIX D: MATLAB CODE FOR NUMERICAL DERIVATIVE CALCULATION IN BIANN

File name: derivative_BIANN.m

```
function [ derivative ] = derivative_BIANN( input, network_number,
time )
%network_number = 1 for model; 2 for critic, 3 for action

delta=1e-8;
N=size(input);
%compute BIANN output with no input tweaking
output=BIANN(input, 0, network_number, 0, time);
%tweak input and calculate derivative
for i=1:N
    input_temp=input;
    %tweak one input
    input_temp(i)=input(i)+delta;
    %compute output after tweaking
    output_temp=BIANN(input_temp, 0, network_number, 0, time);
    %compute derivative
    derivative(i, :)=(output_temp-output)./delta;
end

end
```

BIBLIOGRAPHY

[1] S. Haykin, *Neural Networks - A Comprehensive Foundation (2nd Ed.)*. Upper Saddle River, NJ: Prentice - Hall, 1998.

[2] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* Washington, D.C.: Spartan Books, 1962.

[3] J. E. Dayhoff, *Neural Network Architecture: an Introduction*. New York, NY: Van Nostrand Reinhold Co., 1990, ISBN: 0-442-20744-1.

[4] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric Environment,* vol. 32, pp. 2627-2636, 1998.

[5] J. Park, "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation,* vol. 3, pp. 246-257, 1991.

[6] D. F. Specht, "A General Regression Neural Network," *IEEE Trans. on Neural Networks,* vol. 2, pp. 568-576, 1991.

[7] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE,* vol. 78, pp. 1464-1480, 1990.

[8] T. Kohonen, *Learning vector quantization. In: M.A. Arbib, editor, The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press, 1995, ISBN: 0-262--01148-4.

[9] D. F. Specht, "Probabilistic Neural Networks," *Neural Networks,* vol. 3, pp. 109-118, 1990.

[10] J. L. Elman, "Finding Structure in Time," *Cognitive Science,* vol. 14, pp. 179-211, 1990.

[11] H. Jaeger, "The "Echo State" Approach to Analysing and Training Recurrent Neural Networks," GMD Report 148 - German National Research Institute for Computer Science2001.

[12] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA,* vol. 79, pp. 2554-2558, 1982.

[13] G. A. Carpenter and S. Grossberg, *Adaptive Resonance Theory, In Michael A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks (2nd Ed.)*. Cambridge, MA: MIT Press, 2003, ISBN: 0-262-01197-2.

[14] K. Gopalsamy and X.-Z. He, "Delay-independent stability in bidirectional associative memory networks," *IEEE Transactions on Neural Networks,* vol. 5, pp. 998-1002, 1994.

[15] R. Caruana and A. Nicules-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning (ICML'06)*, New York, NY, USA, 2006, pp. 161-168.

[16] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* vol. 78, pp. 1550-1560, 1990.

[17] G. Hinton and T. J. Sejnowski, *Unsupervised Learning: Foundations of Neural Computation*. Cambridge, MA: MIT Press, 1999.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[19] J. Eckmann*, et al.*, "The Physics of Living Neural Networks," *Physics Reports,* vol. 449, pp. 54-76, 2007.

[20] M. Taketani and M. Baudry, *Advances in Network Electrophysiology: Using Multi-electrode Arrays*. New York, NY: Springer, 2006.

[21] D. J. Bakkum*, et al.*, "Spatio-temporal electrical stimuli shape behavior of an embodied cortical network in a goal-directed learning task " *Journal of Neural Engineering,* vol. 5, pp. 310-323, 2008.

[22] D. J. Bakkum, *et al.*, "Removing Some "A" from AI: Enbodied Culture Networks," *Embodied Artificial Intelligence,* vol. 3139, pp. 130-145, 2004.

[23] A. Novellino, *et al.*, "Connecting Neurons to a Mobile Robot: an in Vitro Bidirectional Neural Interface," *Comput Intell Neurosci.,* 2007.

[24] T. B. DeMarse, *et al.*, "The neurally controlled animat: biological brains acting with simulated bodies," *Autonomous Robots,* vol. 11, pp. 305-310, 2001.

[25] R. Granger, "Engine of the Brain: the Computational INstruction Set of Human Cognition," *AI Magazine,* vol. 27, pp. 15-32, 2006.

[26] Z. A. Bashir and M. E. El-Hawary, "Applying Wavelets to Short-Term Load Forecasting Using PSO-Based Neural Networks," *IEEE Transactions on Power Systems,* vol. 24, pp. 20-27, 2009.

[27] V. H. Ferreira and A. P. Alves da Silva, "Toward Estimating Autonomous Neural Network-Based Electric Load Forecasters," *IEEE Transactions on Power Systems,* vol. 22, pp. 1554-1562, 2007.

[28] H. S. Hippert, *et al.*, "Neural networks for short-term load forecasting: a review and evaluation," *IEEE Transactions on Power Systems,* vol. 16, pp. 44-55, 2001.

[29] K. Methaprayoon, *et al.*, "Multistage Artificial Neural Network Short-Term Load Forecasting Engine With Front-End Weather Forecast," *IEEE Transactions on Industry Applications,* vol. 43, pp. 1410-1416, 2007.

[30] L. Mohan Saini and M. Kumar Soni, "Artificial neural network-based peak load forecasting using conjugate gradient methods," *IEEE Transactions on Power Systems,* vol. 17, pp. 907-912, 2002.

[31] K. Nose-Filho, *et al.*, "Short-Term Multinodal Load Forecasting Using a Modified General Regression Neural Network," *IEEE Transactions on Power Delivery,* vol. 26, pp. 2862-2869, 2011.

[32] S. Qingsong, *et al.*, "Hourly electric load forecasting algorithm based on echo state neural network," in *Control and Decision Conference (CCDC), 2011 Chinese*, 2011, pp. 3893-3897.

[33] T. Saksornchai, *et al.*, "Improve the unit commitment scheduling by using the neural-network-based short-term load forecasting," *IEEE Transactions on Industry Applications,* vol. 41, pp. 169-179, 2005.

[34] T. Senjyu, *et al.*, "One-hour-ahead load forecasting using neural network," *IEEE Transactions on Power Systems,* vol. 17, pp. 113-118, 2002.

[35] H. Showkati, *et al.*, "Short Term Load Forecasting using Echo State Networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN'10)*, 2010, pp. 1-5.

[36] J. W. Taylor and R. Buizza, "Neural network load forecasting with weather ensemble predictions," *IEEE Transactions on Power Systems,* vol. 17, pp. 626-632, 2002.

[37] C. Ying, *et al.*, "Short-Term Load Forecasting: Similar Day-Based Wavelet Neural Networks," *IEEE Transactions on Power Systems,* vol. 25, pp. 322-330, 2010.

[38] G. Zwe-Lee, "Wavelet-based neural network for power disturbance recognition and classification," *IEEE Transactions on Power Delivery,* vol. 19, pp. 1560-1568, 2004.

[39] S. Mishra, *et al.*, "Detection and Classification of Power Quality Disturbances Using S-Transform and Probabilistic Neural Network," *IEEE Transactions on Power Delivery,* vol. 23, pp. 280-287, 2008.

[40] A. G. Bahbah and A. A. Girgis, "New method for generators' angles and angular velocities prediction for transient stability assessment of multimachine power systems using recurrent artificial neural network," *IEEE Transactions on Power Systems,* vol. 19, pp. 1015-1022, 2004.

[41] N. Amjady and S. F. Majedi, "Transient Stability Prediction by a Hybrid Intelligent System," *IEEE Transactions on Power Systems,* vol. 22, pp. 1275-1283, 2007.

[42] Z. Qin, *et al.*, "Application of artificial neural networks in power system security and vulnerability assessment," *IEEE Transactions on Power Systems,* vol. 9, pp. 525-532, 1994.

[43] D. Q. Zhou, *et al.*, "Online Monitoring of Voltage Stability Margin Using an Artificial Neural Network," *IEEE Transactions on Power Systems,* vol. 25, pp. 1566-1574, 2010.

[44] Y. Yi, *et al.*, "Adaptive Echo State Network to maximize overhead power line dynamic thermal rating," in *Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE*, 2009, pp. 2247-2254.

[45] Y. Yi, *et al.*, "Thermal modeling and real time overload capacity prediction of overhead power lines," in *Diagnostics for Electric Machines, Power Electronics and Drives, 2009. SDEMPED 2009. IEEE International Symposium on*, 2009, pp. 1-7.

[46] Y. Yi, *et al.*, "Real-time dynamic thermal rating evaluation of overhead power lines based on online adaptation of Echo State Networks," in *Energy Conversion Congress and Exposition (ECCE), 2010 IEEE*, 2010, pp. 3638-3645.

[47] G. W. Chang, *et al.*, "Radial-Basis-Function-Based Neural Network for Harmonic Detection," *IEEE Transactions on Industrial Electronics,* vol. 57, pp. 2171-2179, 2010.

[48] H. Chao-Ming and W. Fu-Lu, "An RBF Network With OLS and EPSO Algorithms for Real-Time Power Dispatch," *IEEE Transactions on Power Systems,* vol. 22, pp. 96-104, 2007.

[49] V. J. Gutierrez-Martinez, *et al.*, "Neural-Network Security-Boundary Constrained Optimal Power Flow," *IEEE Transactions on Power Systems,* vol. 26, pp. 63-72, 2011.

[50] C. T. Hsu, *et al.*, "Design of adaptive load shedding by artificial neural networks," *IEE Proceedings of Generation, Transmission and Distribution,* vol. 152, pp. 415-421, 2005.

[51] E. J. Thalassinakis, *et al.*, "Method Combining ANNs and Monte Carlo Simulation for the Selection of the Load Shedding Protection Strategies in Autonomous Power Systems," *IEEE Transactions on Power Systems,* vol. 21, pp. 1574-1582, 2006.

[52] L. Chao, *et al.*, "Direct Heuristic Dynamic Programming for Damping Oscillations in a Large Power System," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,* vol. 38, pp. 1008-1013, 2008.

[53] S. Mehraeen, *et al.*, "Power System Stabilization Using Adaptive Neural Network-Based Dynamic Surface Control," *IEEE Transactions on Power Systems,* vol. 26, pp. 669-680, 2011.

[54] Q. Wei, *et al.*, "Coordinated Reactive Power Control of a Large Wind Farm and a STATCOM Using Heuristic Dynamic Programming," *IEEE Transactions on Energy Conversion,* vol. 24, pp. 493-503, 2009.

[55] D. K. Chaturvedi and O. P. Malik, "Generalized neuron-based adaptive PSS for multimachine environment," *IEEE Transactions on Power Systems,* vol. 20, pp. 358-366, 2005.

[56] S. Mishra, "Neural-network-based adaptive UPFC for improving transient stability performance of power system," *IEEE Transactions on Neural Networks,* vol. 17, pp. 461-470, 2006.

[57] T. T. Nguyen and R. Gianto, "Neural networks for adaptive control coordination of PSSs and FACTS devices in multimachine power system," *Generation, Transmission & Distribution, IET,* vol. 2, pp. 355-372, 2008.

[58] G. Gurrala, *et al.*, "Single network adaptive critic design for power system stabilisers," *Generation, Transmission & Distribution, IET,* vol. 3, pp. 850-858, 2009.

[59] M. C. Mabel and E. Fernandez, "Estimation of Energy Yield From Wind Farms Using Artificial Neural Networks," *IEEE Transactions on Energy Conversion,* vol. 24, pp. 459-464, 2009.

[60] Q. Wei, "Echo-state-network-based real-time wind speed estimation for wind power generation," in *International Joint Conference on Neural Networks (IJCNN'09)* 2009, pp. 2572-2579.

[61] A. A. Ferreira, *et al.*, "Investigating the use of Reservoir Computing for forecasting the hourly wind speed in short -term," in *IEEE World Congress on Computational Intelligence (IJCNN'08.)*, 2008, pp. 1649-1656.

[62] T. G. Barbounis, *et al.*, "Long-term wind speed and power forecasting using local recurrent neural network models," *IEEE Transactions on Energy Conversion,* vol. 21, pp. 273-284, 2006.

[63] L. Whei-Min and H. Chih-Ming, "A New Elman Neural Network-Based Control Algorithm for Adjustable-Pitch Variable-Speed Wind-Energy Conversion Systems," *IEEE Transactions on Power Electronics,* vol. 26, pp. 473-481, 2011.

[64] S. Kelouwani and K. Agbossou, "Nonlinear model identification of wind turbine with a neural network," *IEEE Transactions on Energy Conversion,* vol. 19, pp. 607-612, 2004.

[65] L. Hui, *et al.*, "A Stochastic Digital Implementation of a Neural Network Controller for Small Wind Turbine Systems," *IEEE Transactions on Power Electronics,* vol. 21, pp. 1502-1507, 2006.

[66] L. Shuhui, *et al.*, "Using neural networks to estimate wind turbine power generation," *IEEE Transactions on Energy Conversion,* vol. 16, pp. 276-282, 2001.

[67] H. Jaeger, " Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. GMD Report 159," German National Research Center for Information Technology, 2002.

[68] W. Maass, *et al.*, "Real-Time ComputingWithout Stable States:A New Framework for Neural Computation Based on Perturbations," *Neural Computation,* vol. 14, pp. 2531-2560, 2002.

[69] T. K. A. Rahman and G. B. Jasmon, "A new technique for voltage stability analysis in a power system and improved loadflow algorithm for distribution network," in *Energy Management and Power Delivery, 1995. Proceedings of EMPD '95., 1995 International Conference on*, 1995, pp. 714-719 vol.2.

[70] K. J. Makasa and G. K. Venayagamoorthy, "On-line voltage stability load index estimation based on PMU measurements," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1-6.

[71] M. Salmen and P. G. Ploger, "Echo State Networks used for Motor Control," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 1953-1958.

[72] G. K. Venayagamoorthy, "Online design of an echo state network based wide area monitor for a multimachine power system," *Neural Networks,* vol. 20, pp. 404-413, 2007.

[73] S. M. Potter, "What can Artificial Intelligence get from Neuroscience? ," in *Artificial Intelligence Festschrift: The next 50 years*, M. Lungarella*, et al.*, Eds., ed Berlin: Springer-Verlag, 2007, pp. 174-185.

[74] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single neurons, Populations, Plasticity*. New York: Cambridge University Press, 2002.

[75] C. Johnson*, et al.*, "A reversibility analysis of encoding methods for spiking neural networks," in *The 2011 International Joint Conference on Neural Networks (IJCNN),* San Jose, CA, July 31-August 5, 2011, pp. 1802-1809.

[76] C. Johnson and G. K. Venayagamoorthy, "Encoding real values into polychronous spiking networks," in *IEEE World Congress on Computational Intelligence*, Barcelona, Spain, July 18-23, 2010, pp. 1-7.

[77] S. Roychowdhury, "MS Thesis: Spiking Neural Networks Encoding and Decoding Algorithms for Time Series Estimation and System Identification," Missouri University of Science and Technology, May 2012.

[78] C. Johnson, "PhD Dissertation: Spiking Neural Networks and Their Applications," Missouri University of Science and Technology, December 2011.

[79] W. R. Softky, "Simple codes versus effcient codes," *Curr. Opin. Neurobiol.,* vol. 5, pp. 239-247, 1995.

[80] M. N. Shadlen and W. T. Newsome, "Is there a signal in the noise?," *Curr. Opin. Neurobiol.,* vol. 5, pp. 248-250, 1995.

[81] M. N. Shadlen and W. T. Newsome, "The variable discharge of cortical neurons: Implications for connectivity, computation, and information coding," *J. Neurosci.,* vol. 18, pp. 3870–3896, 1998.

[82] J. Gautrais and S. J. Thorpe, "Rate coding vs temporal order coding: Atheoretical approach " *Biosystems,* vol. 48, pp. 57-65, 1998.

[83] E. D. Adrian and Y. Zotterman, "The impulses produced by sensory nerve endings: Part II: The response of a single end organ," *J. Physiol.,* vol. 61, pp. 151-171, 1926.

[84] W. Gerstner, "Neural codes: Firing rates and beyond," in *Proceedings of the National Academy of Sciences of the United States of America*, 1997, pp. 12740-12741.

[85] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.,* vol. 160, pp. 106-154, 1962.

[86] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press, 2001.

[87] S. Thorpe*, et al.*, "Speed of processing in the human visual system," *Nature,* vol. 381, pp. 520-522, 1996.

[88] M. A. Montemurro*, et al.*, "Phase-of-Firing Coding of Natural Visual Stimuli in Primary Visual Cortex " *Current Biology,* vol. 18, pp. 375-380, 2008.

[89] F. Theunissen and J. P. Miller, "Themporal encoding in nervous systems: a rigorous definition," *J. Comput. Neurosci.,* vol. 2, pp. 149-162, 1995.

[90] E. M. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?," *IEEE Transactions on Neural Networks,* vol. 15, pp. 1063-1070, 2004.

[91] C. Koch and I. Segev, *Methods in Neuronal Modeling (2nd Ed.)*. Cambridge, MA: Massachusetts Institute of Technology, 1998, ISBN: 0-262-11231-0.

[92] G. D. Smith*, et al.*, "Fourier analysis of sinusoidally driven thalamocortical realy neurons and minimal integrate-and-fire-or-burst model," *Journal of Neurophysiol.,* vol. 83, pp. 588-610, 2000.

[93] E. M. Izhikevich, "Resonate-and-fire neurons," *Neural Networks,* vol. 14, pp. 883-894, 2001.

[94] G. B. Ermentrout and N. Kopell, "Parabolic bursting in an excitable system coupled with a slow oscillation," *J. Appl. Math.,,* vol. 46, pp. 233-253, 1986.

[95] G. B. Ermentrout, "Type I membranes, phase resetting curves, and synchrony," *Neural Computation,* vol. 8, pp. 979-1001, 1996.

[96] P. E. Latham*, et al.*, "Intrinsic dynamics in neuronal networks. I. Theory," *Journal of Neurophysiol.,* vol. 83, pp. 808-827, 2000.

[97] R. FitzHugh, "Impulses and physiological states in models of nerve membrane," *Biophys. J.,* vol. 1, pp. 445-466, 1961.

[98] R. M. Rose and J. L. Hindmarsh, "The assembly of ionic currents in a thalamic neuron. I The three-dimensional model," in *Proc. R. Soc. Lond. B*, 1989, pp. 267–288.

[99] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber " *Biophys. J.,* vol. 35, pp. 193–213, 1981.

[100] H. R. Wilson, "Simplified dynamics of human and mammalian neocortical neurons," *J. Theor. Biol.,* vol. 200, pp. 375–388, 1999.

[101] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and application to conduction and excitation in nerve," *J. Physiol.,* vol. 117, pp. 500–544, 1954.

[102] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Networks,* vol. 14, pp. 1569–1572, 2003.

[103] J. Mazumdar*, et al.*, "Neural Network Based Method for Predicting Nonlinear Load Harmonics," *IEEE Transactions on Power Electronics,* vol. 22, pp. 1036-1045, 2007.

[104] S. Rahmani*, et al.*, "A New Control Technique for Three-Phase Shunt Hybrid Power Filter," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 2904-2915, 2009.

[105] B. Singh and J. Solanki, "An Implementation of an Adaptive Control Algorithm for a Three-Phase Shunt Active Filter," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 2811-2820, 2009.

[106] F. D. Freijedo*, et al.*, "A Signal-Processing Adaptive Algorithm for Selective Current Harmonic Cancellation in Active Power Filters," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 2829-2840, 2009.

[107] S. Kuo-Kai*, et al.*, "Model Reference Adaptive Control Design for a Shunt Active-Power-Filter System," *IEEE Transactions on Industrial Electronics,* vol. 55, pp. 97-106, 2008.

[108] M. A. M. Radzi and N. A. Rahim, "Neural Network and Bandless Hysteresis Approach to Control Switched Capacitor Active Power Filter for Reduction of Harmonics," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 1477-1484, 2009.

[109] M. Cirrincione*, et al.*, "A Single-Phase DG Generation Unit With Shunt Active Power Filter Capability by Adaptive Neural Filtering," *IEEE Transactions on Industrial Electronics,* vol. 55, pp. 2093-2110, 2008.

[110] M. Cirrincione*, et al.*, "Current Harmonic Compensation by a Single-Phase Shunt Active Power Filter Controlled by Adaptive Neural Filtering," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 3128-3143, 2009.

[111] K. Drobnic*, et al.*, "Predictive Direct Control Applied to AC Drives and Active Power Filter," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 1884-1893, 2009.

[112] J. Mazumdar and R. G. Harley, "Recurrent Neural Networks Trained With Backpropagation Through Time Algorithm to Estimate Nonlinear Load Harmonic Currents," *IEEE Transactions on Industrial Electronics,* vol. 55, pp. 3484-3491, 2008.

[113] J. Mazumdar and R. G. Harley, "Utilization of Echo State Networks for Differentiating Source and Nonlinear Load Harmonics in the Utility Network," *IEEE Transactions on Power Electronics,* vol. 23, pp. 2738-2745, 2008.

[114] X. Peng*, et al.*, "Seven-Level Shunt Active Power Filter for High-Power Drive Systems," *IEEE Transactions on Power Electronics,* vol. 24, pp. 6-13, 2009.

[115] O. Vodyakho*, et al.*, "Novel Direct Current-Space-Vector Controlfor Shunt Active Power Filters Basedon the Three-Level Inverter," *IEEE Transactions on Power Electronics,* vol. 23, pp. 1668-1678, 2008.

[116] R. Kuffel*, et al.*, "RTDS-a fully digital power system simulator operating in real time," in *WESCANEX 95. Communications, Power, and Computing. Conference Proceedings. IEEE*, 1995, pp. 300-305 vol.2.

[117] E. M. Izhikevich, "Polychronization: Computation with Spikes," *Neural Computation,* vol. 18, pp. 245-282, 2006.

[118] N. Caporale and D. Yang, "Spike-Timing–Dependent Plasticity: A Hebbian Learning Rule," *Annual Review of Neuroscience,* vol. 31, pp. 25-46, 2008.

[119] P. Kundur, *Power system stability and control* New York: McGraw-Hill, 1994, ISBN:978-0070359581.

[120] P. Jung-Wook*, et al.*, "Adaptive-critic-based optimal neurocontrol for synchronous generators in a power system using MLP/RBF neural networks," *IEEE Transactions on Industry Applications,* vol. 39, pp. 1529-1540, 2003.

[121] D. V. Prokhorov and D. C. Wunsch, II, "Adaptive critic designs," *IEEE Transactions on Neural Networks,* vol. 8, pp. 997-1007, 1997.

[122] S. M. Potter*, et al.*, "Closing the Loop: Stimulation Feedback Systems for Embodied MEA Cultures," in *Advances in Network Electrophysiology Using Multi-Electrode Arrays*, M. Taketani and M. Baudry, Eds., ed New York: Springer, 2006, pp. 215-242.

VITA

Jing Dai was born in China in 1983. She received a Bachelor of Engineering degree from Tsinghua University, Beijing, China, in 2005 and a Master of Engineering degree from Georgia Institute of Technology at Atlanta, Georgia, in 2009, both in Electrical Engineering.

Since August 2007, she has been working computational intelligence and its application in the power system, as a graduate Research Assistant in the electric power group of the Georgia Institute of Technology.

Her research interests include computational intelligence, biologically-inspired artificial neural networks and optimal control applications in power systems. She has published 7 papers in refereed journals and international conference proceedings in these areas.